CHAPTER 3 SYSTEM ANALYSIS

3.1 Performance Requirements

Any strong authentication protocols are considered to have all of the following characteristics. E-PAP which is discussed in this dissertation has met all goals and have other desirable characteristics which is explained later.

3.1.1 Provide Mutual Authentication

Unilateral authentication is an entity authentication that provides one entity with assurance of the other's identity but not vice versa, such as a log on in a telnet system. To provide strong authentication, unilateral authentication is not enough. Mutual authentication is desirable, which provides both entities with assurance of each other's identity. Besides, it must also prove to each of two parties that the other knows the password. For a better system, it should achieve zero-knowledge mutual authentication, which means prove that each entity knows a small secret (password), without revealing it to each other.

3.1.2 Prevent On-Line Dictionary and Brute-Force Attack

On-line dictionary can be easily detected by counting access failures. A system may log any accesses from registered users, anonymous users or others that successfully or unsuccessfully provide a correct password.

3.1.3 Prevent Off-Line Dictionary and Brute-Force Attack

Off-line dictionary attack and brute-force attacks presents a more complex threat. These attacks can be made by someone posing as a legitimate entity to gather information, or by one who monitors the messages between two parties during a legitimate valid exchange. Even tiny amounts of information "leaked" during an exchange can be exploited. The method must be immune to such off-line attack, even for tiny passwords [39].

3.1.4 Integrated Key Exchange

Integrated key exchange is desired because if separate steps of authentication and key exchange will create opportunities for an attacker in the middle. Strong key exchange requires the participation of both parties, and should be an integral part of the process [39].

3.1.5 No Persistent Recorded Secret or Sensitive Host-Specific Data

No persistent recorded data means the user needs no additional symmetric, public, or private keys, and this will simplify the user's side of the system as well as make the password an independent factor. If a system requires persistent that data be generated, distributed, and securely stored, this will pose additional problems. The same problem will also occurr if a system requires specially protected memory where it weakens the security model by adding another point of failure. Secret data must never be revealed, and non-secret sensitive data must be kept tamper-proof. Systems where the security of a password depends on a stored key are easily constructed, but they just move the basis of security from the password to the key. If the key is stolen, the password can be compromised. By eliminating persistent user-keys, this will also remove the above problems [39].

3.1.6 Forward Secrecy

Forward secrecy means revealing the password to an attacker does not help him to obtain any information such as session keys of past sessions. A stolen session key also does not help an attacker to carry out a brute-force attack on the password [5].

3.2 Obsolete Password Method

After examining strong authentication requirements, this section investigates some obsolete password methods.

3.2.1 Clear password

Clear password is still predominant in the Internet today, as in telnet password, ftp password, Basic Access Authentication and any password sent in an unencrypted session. The protocol is simple: Alice (the user or client) sends Bob (the host or server) her username and her plaintext password. Bob verifies the password either by comparing it directly to his version of Alice's password or applying a function (normally one-way function) first and checking it against a database of stored hashes. These are vulnerable to easy sniffing attacks.

An example of clear password method is Basic Access Authentication scheme [10] that is used in "HTTP/1.0" [20]. Typically, it requires a username and a password for authentication. Files on the server contain a list of users, passwords and groups, passwords are stored in an encrypted form. When the server receives a request for a document that is protected by basic authentication, it sends an "authentication required" message back to the client. The browser then prompts the user for a name and a password in a dialog box (Figure 3-1). The client resubmits the request with the username: password in the authentication line. Basic Access Authentication scheme is not considered as a secure method of user authentication as the user name and password are passed over the network in clear text. Therefore they are easily "sniffed".

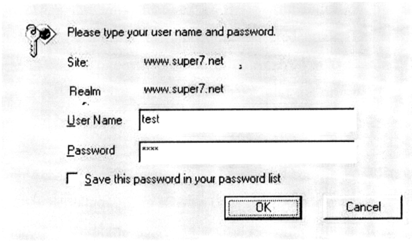

Please type your user name and password.
Site: www.super7.net
Realm www.super7.net
User Name test
Password xxxx
Save this password in your password list

Figure 3-1: Basic Authentication

3.2.2 Scrambled password

This method simply obscure the password, using some well-known or easily discoverable method. They are not significantly stronger than a clear password.

3.2.3 Challenge/Hashed-Random Response (CHRAP)

Many variations of hash-based challenge response schemes have been used since the early 1980s. Some are better than others, and all are better than clear-text password and scrambled password. However, all CHRAP methods are open to dictionary or brute-force attack when the password is small or not-so-random. This approach has been used in RPC (remote procedure control) based system like NetWare 3, Banyan VINES, Microsoft's LAN Manager for Windows NT4, CRAM (Challenge-Response Authentication Mechanism), PPP CHAP (Challenge-Handshake Authentication Protocol), and HTTP Digest authentication for HTTP/1.1.

The Digest Access Authentication scheme [10] which is used in "HTTP/1.1" [21] and supported by Microsoft Internet Information Server (IIS) version 5 in Windows 2000 is not intended for the need for security in the World Wide Web. Digest Authentication offers the same features as Basic Authentication but involves a different method for transmitting the authentication credentials. The authentication credentials pass through a one-way process, often referred to as hashing. The result of this process is called a hash, or message digest, and the original text cannot be deciphered from the hash [11].

Like the Basic Access Authentication, the Digest scheme is based on a simple challenge-response paradigm. This scheme does not provide encryption of message content. The intent is simply to create an access authentication method that avoids the most serious flaws of Basic Authentication. Below are attacks that have been considered for digest authentication [36]:

- Person-in-the-middle

A person in between the server and the client cannot forge a request for another object or modify the request in any significant manner. However, it is currently possible for a person in the middle to modify the reply since these are not currently signed [36].

- Analysis of password file leading to compromised security
  The security of this mechanism depends entirely on the security of the password file, as this mechanism will generate a password file that is plaintext equivalent to password. If a third party reads this then access is effectively granted even though the password itself is safe [36].

3.2.4 Kerberos Logon

Like other general failure of many obsolete password methods, Kerberos logon presumes that passwords have to be large. Here's a simple example of a bad way for Alice to verify that Bob knows a small password, *S*. Alice sends a random nonce *R* to Bob, and Bob returns $Q = h(R, S)$ to prove that he knows *S*. *h()* is a one-way hash of the nonce combined with the password, which is stored in `/etc/passwd` (for UNIX system) as a password file which is plaintext-equivalent (Section 2.2.2). But because the space is searchable with brute force, an eavesdropper can perform a dictionary attack by repeatedly computing $h(R, Si)$ for each candidate password *Si* and compares the result to *Q*. A possible solution is to restrict access to the file `/etc/passwd`.

In Kerberos V4 and V5, a password encrypts an initial ticket. The data contained in the ticket allows a "verifiable text" dictionary attack [12]. The V4 method is even weaker than challenge-response, since a dictionary attack can be performed by anyone who simply asks for a Ticket-Granted-Ticket (TGT) at any time. Even with the addition of timestamp-based pre-authentication in Kerberos V5, which is also used in Windows 2000, the protocol is still vulnerable to dictionary and brute-force attack by an eavesdropper. The key distribution server encrypts its initial response packet using a key derived from a user's password. An attacker may record such a packet and attempt to decrypt it using keys derived from a series of guesses as to the password. The attacker is able to determine whether his guess is correct or not as a correct guess will produce recognizable data, such as the name of a network service or the time of day.

3.2.5 S/Key

There are two sides to the operation of the S/KEY one-time password system. On the client side, the appropriate one-time password must be generated. On the host side, the server must verify the one-time password and permit the secure changing of the user's secret pass-phrase.

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, the number of applications is reduced by one. 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. This technique was first suggested by Leslie Lamport [44].

Implementation of the S/Key scheme would break the idempotence of the HTTP protocol. In addition, problems arise when a client sends multiple requests to the same server, which are processed, out of order. Another difficulty is that the shared secret can only be used a limited number of times. In a session-based protocol such as telnet this is not a problem since logins are infrequent. In the context of HTTP/1.0 this is a major disadvantage since a user may perform many hundred operations a day thus requiring key changes on a weekly or even daily basis [36].

Concern should also be raised about the applicability of certain hash methods with the S/Key scheme, if the hash function should contain a strange attractor to either a fixed point or a cycle of fixed points, an attack might be possible. The mathematical property of recursive application of hash functions for very high orders is not known. Such a system should certainly be considered with caution.

S/Key addresses a different problem, in that it can be used in conjunction with programs originally designed to accept only clear-text password. But S/Key alone cannot solve the eavesdropper dictionary attack against small password.

3.2.6 SecurID

SecurID (Figure 3-2) is a two-factor authentication system, which includes something you know and something you have, developed and sold by Security Dynamics [41]. It is generally used to secure either local or remote access to computer networks. Each SecurID user has a memorized PIN or password, and a hand-held token with a LCD display. The token displays a new pseudorandom value, called the tokencode, at a fixed time interval, usually one minute. The user combines the memorized factor with the tokencode, either by simple concatenation or entry on an optional keypad on the token, to create the passcode, which is then entered to gain access to the protected resource.

Figure 3-2: SecurID

The SecurID token is a battery powered, hand-held device containing a dedicated microcontroller. The microcontroller stores in RAM, the current time, and a 64-bit seed value that is unique to a particular token. At the specified interval, the seed value and the time are combined through a proprietary algorithm stored in the microcontroller's ROM, to create the tokencode value.

An authentication server verifies the passcodes. The server maintains a database, which contains the seed value for each token, and the PIN or password for each user. From this information, and the current time, the server generates a set of valid passcodes for the user and checks each one against the entered value.

SecurID and most two-factor systems were not designed to solve the small password problem. In this case, the security of the system relies on the secrecy of a key stored in the SecurID card. It also relies on a memorized password, in case the card is stolen. While the small password does help verify a human presence, the password method remains a weak link. Using a stolen card, an informed attacker who has monitored a SecurID session with the card can mount an attack on the password and use the card himself.

3.2.7 Clear Password Over An SSL Signed Channel

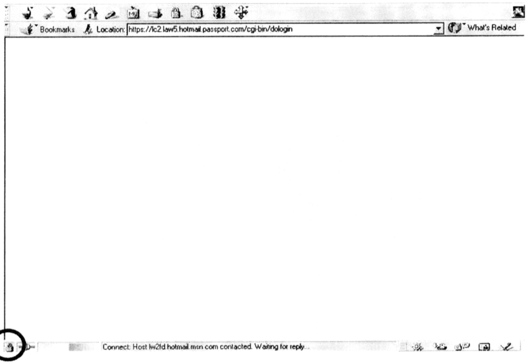
The use of digital signatures in web browsers has created a new style of authentication that is becoming widely used. But the one-way nature of digital signatures enables a variety of identity or "name spoofing" attacks. One should recognize that this is still just a stored-key method authenticating two machines. It does not verify a human presence. Sending a clear password over an SSL connection adds to the threat of name spoofing attacks and cannot provide reliable prove that the right Alice is talking to the right Bob [33].

Besides that, authentication is needed at the application level in the long-term. SSL also requires a new proxy protocol because of current proxy implementation (Version 1.1) has a weak routine for seeding random number generator which creates session keys that are guessable. User may also need to take care when sending clear password over SSL channel where

- User might not check SSL icon (Figure 3-3).
- User might not check certificate. The certificate may have expired or is not yet valid (Figure 3-4).
- User might not notice a misspelled name of URL that may cause server-spoofing attacks.

- Navigator 3.0 and Communicator 4.x have an option to allow on-disk caching of data fetched over SSL connections.

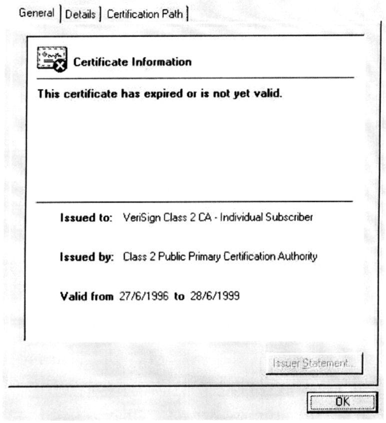
Furthermore, there are several other network layer security proposals like Microsoft's Private Communication Technology (PCT) or Secure HyperText Transfer Protocol (SHTTP) [32] which are not compatible with each other.



Location: https://lc2.law5.hotmail.passport.com/cgi-bin/dologin

Connect: Host lw2fd.hotmail.msn.com contacted. Waiting for reply...

Figure 3-3: SSL icon for Netscape Communicator



Certificate Information

This certificate has expired or is not yet valid.

Issued to: VeriSign Class 2 CA - Individual Subscriber

Issued by: Class 2 Public Primary Certification Authority

Valid from 27/6/1996 to 28/6/1999

Figure 3-4: An expired certificate for Internet Explorer 5

An SSL server certificate contains two pieces of data of potential security interest: the name of the keyholder (usually a corporate name) and the DNS name for the server. There are authorities on DNS name assignments, but none of the SSL CAs listed in the popular browsers is such an authority. That means that the DNS name in the certificate is not an authoritative statement. There are authorities on corporate names. These names need to be registered when one gets a business license. However, none of the SSL CAs listed in the browsers is such an authority. In addition, when some server holds an SSL server certificate, it has permission to do SSL. Who granted the authority to an SSL CA to control that permission? Is the control of that permission even necessary? It serves an economic purpose (generating an income stream for CAs) but does it serve a security purpose? What harm is done if an uncertified server were allowed to use encryption? The answer is none [33].

Does the proper use of these certificates require user actions? Do users perform those actions? For example, when a user establishes an SSL connection with his browser, there's a visual indication that the SSL protocol worked and the link is encrypted. But whom is the user talking securely with? Unless the user take the time to read the certificate that he received [33].

3.3 Key-Exchange Algorithms

After discussing obsolete password methods, this section discusses some key-exchange algorithm, which is core of E-PAP System.

3.3.1 Diffie-Hellman

The Diffie-Hellman (DH) key agreement protocol (also called exponential key agreement) was developed by Diffie and Hellman in 1976. DH was the first public-key algorithm ever invented. The protocol allows two users to exchange a secret key over an insecure medium without any prior secrets.

The math is simple. First, Alice and Bob agree on a large prime number p and g (usually called a generator), which is an integer less than p, with the following property: for every number n between 1 and p-1 inclusive, there is a power k of g such that $g^k = n \bmod p$.

Then, the protocol goes as follow:

Notation:

x, y = random large integer

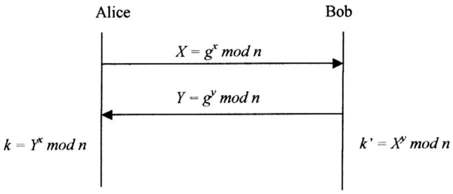


Figure 3-5: Diffie-Hellman

Alice and Bob then compute k and k' respectively where both k and k' are equal to $g^{xy} \bmod n$. Eve listening on the channel can not compute that value; she only knows n, g, X and Y. She can not solve the problem unless she can compute the discrete logarithm and recover x or y. Therefore, k is the secret key that both Alice and Bob computed independently.

However, the Diffie-Hellman key exchange by itself does not provide authentication and is vulnerable to a man-in-the-middle attack. In this attack, an opponent Carol intercepts Alice's public value and sends her own public value to Bob. When Bob transmits his public value, Carol substitutes it with her own and sends it to Alice. Carol and Alice thus agree on one shared key and Carol and Bob agree on another shared key. After this exchange, Carol simply decrypts any messages sent out by Alice or Bob, and then reads and possibly modifies them before re-encrypting with the appropriate key and transmitting them to the other party. This vulnerability is present because Diffie-Hellman key exchange does not authenticate the participants. Possible solutions include the use of digital signatures and other variant protocols like Station-to-Station (STS) protocol Encrypted Key Exchange.

In recent years, the original Diffie-Hellman protocol is understood to be an example of a much more general cryptographic technique, the common element being the derivation of a shared secret value (or key) from one party's public key and another party's private key. The parties' key pairs may be generated anew at each run of the protocol, as in the original Diffie-Hellman protocol. The public keys may be certified, so that the parties can be authenticated and there may be a combination of these attributes. The draft ANSI X9.42 illustrates some of these combinations, and a recent paper by Blake-Wilson, Johnson, and Menezes provides some relevant security proofs [7].

3.3.2 Station-to-Station Protocol

The authenticated Diffie-Hellman key agreement protocol, or Station-to-Station (STS) protocol, was developed by Diffie, van Oorschot, and Wiener in 1992 to defeat the man-in-the-middle attack on the Diffie-Hellman key agreement protocol. The immunity is achieved by allowing the two parties to authenticate themselves to each other by the use of digital signatures and public key certificates.

Roughly speaking, the basic idea is as follows. Prior to the execution of the protocol, the two parties Alice and Bob each obtains a public/private key pair and a certificate for the public key. During the protocol, Alice computes a signature on certain messages, covering the public value g a mod p. Bob proceeds in a similar way. Even though Carol is still able to intercept messages between Alice and Bob, she cannot forge signatures without Alice's private key and Bob's private key. Hence, the enhanced protocol defeats the man-in-the-middle attack

3.3.3 Encrypted Key Exchange (EKE)

Encrypted Key Exchange (EKE) protocol was designed by Steve Bellovin and Michael Merritt [14]. It provides security and authentication on computer networks by using both symmetric and public-key cryptography in a novel way. The basic idea is a shared secret key is used to encrypt a randomly generated public key.

Table 3-1: Notation for EKE protocol.

A; B	System principals. (*Alice* and *Bob*).
S	The password: a shared secret, often used as a key.
Y(info)	Symmetric (secret-key) encryption of "*info*" with key *Y*.

Y^{1}(info)	Symmetric (secret-key) decryption of "*info*" with key *Y*.
$E_k(X)$	Asymmetric (public-key) encryption of *X* with (public) key E_k
challenge_A	A random challenge generated by *A*.
challenge_B	A random challenge generated by *B*.
p	A huge prime numbers suitable for Diffie-Hellman
q	A large prime factor of p-1
A→B: m	Alice sends *m* to Bob

Alice and Bob share a common password, *S*. Using this protocol, they can authenticate each other and generate a common session key, *K*.

1. *A* generates a random public key E_A and encrypts it in a symmetric cryptosystem with key *S* to produce $S(E_A)$.

$$A \rightarrow B: A, S(E_A)$$

This message includes her name in the clear.

Sharing the password *S*, B decrypts to obtain E_A, generates a random secret key *R*, and encrypts it in both the asymmetric cryptosystem with key E_A and in the password key to produce $S(E_A(R))$.

$$B \rightarrow A: S(E_A(R))$$

2. *A* decrypts the message to obtain *R*, generates a unique challenge challenge_A and encrypts it with R to produce $R(\text{challenge}_A)$.

$$A \rightarrow B: R(\text{challenge}_A)$$

3. *B* decrypts the message to obtain challenge_A, generates a unique challenge challenge_B and encrypts the two challenges with the secret key *R* to obtain $R(\text{challenge}_A, \text{challenge}_B)$.

$$B \rightarrow A: R(\text{challenge}_A, \text{challenge}_B)$$

4. *A* decrypts to obtain challenge *A* and challenge_B and compares the former against her earlier challenge. If it matches, she encrypts challenge_B with R to obtain.

$$A \rightarrow B: R(\text{challenge}_B)$$

5. If the challenge-response protocol in steps 1 to 5 is successful, logon is successful and the parties proceed with the logon session, using the symmetric cryptosystem and session key *R* for protection.

The challenge-response portion of the protocol, in steps 3 to5 is a standard technique for validating cryptographic keys. (If a party sends challenge *c* encrypted by *R*, where *c* was never used before, and receives another encrypted message containing *c* in reply, it follows that the message originator has the ability to encrypt messages with *R*.) This portion of the protocol could be replaced by other mechanisms for validating *R*. For example, the time could be exchanged encrypted by *R*, under the security-critical assumption that clocks are monotonic and synchronized, as in Kerberos [38].

3.4 Simple Password-Authenticated Exponential Key Exchange

Simple Password-Authenticated Exponential Key Exchange (SPEKE) [39] is a authenticated key exchange. It uses DH key-exchange algorithm to protect the password from off-line dictionary attack and uses a password to prevent standard DH man-in-middle attack as discussed in 3.3.1. The basic idea for SPEKE is two parties who share only a small password *P*, perform mutual authentication over insecure network, such as the Internet, proving to each other their knowledge of *P* and generating a new large session key *K*.

3.4.1 The Protocol

Table 3-2: Notation of SPEKE protocol

S	A small shared password for Alice and Bob.
P	A huge prime number suitable for Diffie-Hellman.
Q	A large prime factor of $p-1$.
G	A suitable DH base, either primitive, or of large prime order.
Z_p^*	The group of integers from 1 to p-1, under multiplication modulo p.

Gx	A subgroup of Z_p* of order x. x is a factor of p-1.
$f(S)$	A function that converts S into a suitable DH base.
R_A, R_B	Random numbers chosen by Alice, and Bob.
Q_A, Q_B	Exponential values sent by Alice, and Bob.
$E_k(m)$	A symmetric encryption function of m using key *k*.
$h(m)$	A strong one-way hash function of *m*.
$A \rightarrow B$: *m*	Alice sends m to Bob.
K	Generated session key.

First Stage

SPEKE has two stages that are integrated together. The first stage uses a DH exchange to establish a shared key *K*, but instead of the commonly used fixed primitive base *g*, a function *f* converts the password *S* into a base for exponentiation. The rest of the first stage is pure Diffie-Hellman, where Alice and Bob start out by choosing two random numbers R_A and R_B:

1. Alice computes: $Q_A = f(S)^{R_A} \bmod p$ $A \rightarrow B: Q_A$
2. Bob computes: $Q_B = f(S)^{R_B} \bmod p$ $B \rightarrow A: Q_B$
3. Alice computes: $K = h(Q_B^{R_A} \bmod p)$
4. Bob computes: $K = h(Q_A^{R_B} \bmod p)$

Second Stage

In the second stage of SPEKE, both Alice and Bob confirm each other's knowledge of *K* before proceeding to use it as a session key. One way is:

5. Alice chooses a random C_A, $A \rightarrow B: E_K(C_A)$
6. Bob chooses a random C_B, $B \rightarrow A: E_K(C_B, C_A)$
7. Alice verifies that C_A is correct, $A \rightarrow B: E_K(C_B)$
8. Bob verifies that C_B is correct.

Alice and Bob only 'belief' each other after step 8 complete.

3.4.2 Why is SPEKE used?

Recently, there are a few cryptographic protocols to negotiate an authenticated cryptographic key based on a small shared secret without revealing anything else about the small shared secret, like Simple Password-Authenticated Exponential Key Exchange (SPEKE) [39], Secure Remote Protocol (SRP) [19], Augmented Encrypted Key Exchange (AEKE) [15] and Open Key Exchange (OKE) [60]. However, this dissertation has chosen SPEKE as implementation of E-PAP because of:

- SPEKE embeds the password in a Diffie-Hellman key exchange, the heart of all public-key methods. By relying on the difficulty of the discrete-log problem, SPEKE rests on the same foundation as all commercial public-key methods. It provides 1000-bit protection for even 20-bit passwords (equal to a password only 3.5 characters).
- There are no articles that can be found from the Internet regarding SPEKE has been broken at this moment.
- A software developer's toolkit, FreeSPEKE SDK is provided by Integrity Sciences Inc., the inventor of SPEKE. This SDK is free for non-commercial, academic and evaluation purposes.
- Entrust® Technologies, a global market and technology leader in Managed PKI solutions, has licensed SPEKE for use in their public key infrastructure (PKI) products [66]. This proves that SPEKE is strong enough for commercial use.