CHAPTER 3

A TEXTUAL PASSWORD BASED AUTHENTICATION SCHEME

Textual Password based Authentication is the most widely used method to verify the validity of a user. Among these methods, Two-Factor Authentication is considered to be more secure than the conventional single factor authentication. Till today, many Two-Factor Authentication Schemes have been proposed both with and without smart cards but each scheme have its own merits and demerits. In the last decade, many researchers focused their work on providing the efficient and secure dynamic ID based mechanisms to authenticate the user without verification table.

The dynamic ID scheme [59] by Das et al. was found to be vulnerable to many attacks. One such attack was shown by Liao et al. [65] who also proposed an enhancement to [59] in the same paper. Their work along with security analysis of their scheme is already discussed in section 2.3.1. This section presents a proposed scheme that provides better security compared to most of the current dynamic ID schemes referred in 2.3.2. It also presents the security analysis of the proposed scheme and compares the efficiency and functionality with Liao et. al scheme. Moreover, section 6.1 presents the proof of concept implementation of the proposed scheme which is believed to be (based on
the survey) the first dynamic ID based scheme to be implemented and tested for security and usability.

3.1 THE PROPOSED SCHEME

This section discusses the proposed scheme designed for web based services. The proposed scheme can be implemented with Smart Cards or USB tokens. It consists of three phases, Registration, Login & Authentication and Password Change phase.

3.1.1 Registration Phase

If a user $U_i$ wants to register with a server, he chooses a password $PW_i$.

*Step R1:* $U_i$ submits $h(ID_i)$ and $h(PW_i)$ to server through secure channel.

*Step R2:* $S$ computes $Q_i = h(h(ID_i) || h(PW_i))$; $N_i = h(PW_i) \oplus h(x \oplus h(ID_i))$, where $x$ is a secret key of the server and $h(.)$ is the one way hash function.

*Step R3:* $S$ personalizes the smart card with the parameters $N_i$, $Q_i$ and $y$; where $y$ is the server’s secret number stored in each registered user’s smart card.

*Step R4:* $S = \rightarrow U_i$: Smart card.

3.1.2 Mutual Authentication & Key Agreement Phase

Whenever a registered user wants to login to access resources at the server, he inserts the smart card into the reader or attaches the USB token to the system and proceeds as follows:
**Step L1:** $U_i$ requests for login.

**Step L2:** Based on the received request, the server generates a random secret ‘$a_1$’ and then computes ‘$A_1$’ as per the DHKE protocol and sends it with ‘g’, ‘r’ and login page.

**Step L3:** $U_i$ keys his identity $ID_i$.

**Step L4:** The client computes $Q_i^* = h(h(ID_i) \parallel h(PW_i))$; checks whether $Q_i^*$ equals $Q_i$ (stored in smart card); If valid it invokes the DHKE protocol which generates a random secret ‘$b_1$’ and then computes ‘$B_1$’ as per the protocol. Using the received value $A_1$ and computed values ‘$b_1$’ and ‘$B_1$’, the protocol computes the secret key ‘$p_i$’.

**Step L5:** The client then computes $CID_i = h(PW_i) \oplus h(N_i \oplus y \oplus p_i)$;

$$Z_i = h(h(CID_i) \oplus h(PW_i))$$

**Step L6:** Computes $C_i = h(p_i \oplus Z_i \oplus y)$

**Step L7:** $D_i = h(ID_i) \oplus h(p_i)$

**Step L8:** $U_i \rightarrow S$: $(CID_i, C_i, D_i, B_1)$

Upon receipt of the login message $(CID_i, C_i, D_i, B_1)$, the server first computes $p_i$ using the received $B_1$ and earlier computed values ‘$a_1$’, $A_1$. It then proceeds as follows:

**Step A1:** $h(ID_i) = D_i \oplus h(p_i)$; $h(PW_i) = CID_i \oplus h(x \oplus h(ID_i)) \oplus y \oplus p_i)$

**Step A2:** $Z_i' = h(h(CID_i) \oplus h(PW_i))$, $C_i' = h(p_i \oplus Z_i' \oplus y)$. It then checks $C_i'$ and $C_i$ are equal. If it holds, the server accepts the login request. Otherwise, rejects the login request.

**Step A3:** $S$ generates a random number $p_2$ and computes $M = h(p_1 + 1)$

**Step A4:** $S \rightarrow U_i$: $(M)$
**Step A5:** Upon receiving the mutual authentication message ‘M’, the Client computes \( M' = h(p_1+1) \) and compares \( M' \) with the received \( M \). If it is equal, it confirms that he is communicating with valid \( S \).

Now both Client and Server Computes the Session Key as

\[
SK = h(h(Z_i) \oplus h(p_1+2)) \quad \text{and} \quad SK = h(h(Z'_i) \oplus h(p_1+2))
\]

respectively.

### 3.1.3 Password Change Phase

A registered user can change his password by sending password change request to the server; but for this, he has to first login to the server. If the login is successful, then both client and server will have the session key to encrypt / decrypt the further communication. So, when the user submits a request for changing password, the client performs the following steps:

**Step C1:** Client generates a random number \( P_k \) and encrypts it with session key as \( E_{Sk}(P_k) \).

**Step C2:** Upon receipt of \( E_{Sk}(P_k) \), server decrypts it as \( D_{Sk}(P_k) \) and checks the freshness of the nonce. If valid it creates \( P_k+1 \).

**Step C3:** Server sends \( E_{Sk}(\text{Page}, P_k+1) \)

**Step C4:** Upon receipt of \( E_{Sk}(\text{Page}, P_k+1) \), the client decrypts it as \( D_{Sk}(\text{Page}, P_k+1) \) and checks the freshness of nonce.

**Step C5:** User submits old \( PW_i \) and new password \( PW_i^* \)

**Step C6:** Client computes \( N_i^* = h(PW_i)^* \oplus h(PW_i) \oplus N_i \);

Replaces \( N_i \) with \( N_i' \).
Step C7: Client updates the contents of smart card as \( \{N_i^*, Q_i, y\} \); it also sends a confirmation message to server.

In password change phase, the client executes all the required computational steps and updates the smart card. Therefore, the computation and communication cost at the server is drastically reduced. Moreover, the user secret parameter \( N_i \) is never transmitted over the channel, providing the security against interception.

The summarized Registration, Mutual Authentication & Key Agreement and Change Password Phases are depicted in Figures 3.1, 3.2 and 3.3 resp.:
**Figure 3.2** Mutual Authentication & Key Agreement Phase

**Client**

- User inserts smart card and keys his ID and Pwi.
- It computes $Q_i^* = h(h(ID_i) | h(Pwi))$ and checks $Q_i^* = Q_i$
- Computes $CID_i = h(PWi) \oplus h(Ni \oplus y \oplus p_i)$;  
  $Z_i = h(h(CID_i) \oplus h(PWi));  
  Di = h(ID_i) \oplus h(p_i)$  
  and $Ci = h(p_i \oplus Z_i \oplus y)$;  
- Generates a secret ‘$b_1$’ and computes $B_1$ as per DHKE algorithm.  
- Computes shared key ‘$p_i$’  
- $M' = h(p_i + 1)$ and compares $M'$ with the received $M$. If not valid, it terminates the session else  
- Computes the session key $SK = h(h(Zi) \oplus h(p_i + 2))$

**Server**

- Generates a secret ‘$a_1$’ and computes $A_1$ as per DHKE algorithm.  
- Computes $p_i$ using $B_1, a_1, & A_1$ and  
  $h(ID_i) = Di \oplus h(p_i)$  
  $h(PWi) = CID_i \oplus h(h(ID_i) \oplus y \oplus p_i)$  
  $Z_i' = h(h(CID_i) \oplus h(PWi))$  
- It then checks whether received $Ci$ equals $h(p_i \oplus Z_i' \oplus y)$. If it holds, it  
  Computes $M = h(p_i + 1)$  
- Computes the Session Key $SK = h(h(Zi') \oplus h(p_i + 2))$
3.2 SECURITY ANALYSIS

The proposed scheme addresses the security weaknesses of Liao et al. scheme and provides better security strength against current dynamic ID based schemes. This section presents the security analysis of the proposed scheme against various attacks and provides comparative analysis of the security of this scheme with Liao et al. scheme.

Security against Impersonation Attack

The impersonation attack discussed in section 2.3.1 does not work here in this scheme because,

In step 2 of verification phase, \( h(PW_i) \) is computed as
h(PW_i) = CID_i \oplus h(x \oplus ID) \oplus y \oplus p_i \) instead of
h(PW_i) = CID_i \oplus h(N_i \oplus y \oplus T)

Since, in the proposed scheme computation of h(PW_i) in step 2 of verification phases does not uses N_i value as compared to step 2 of verification phase of Liao et al., the adversary cannot succeed. Moreover, to perform an impersonation attack the adversary must have the knowledge of server’s secret ‘x’ and smart card secret ‘y’, which are never transmitted through the channel.

**Security against Reflection Attack**

The attack discussed in section 2.3.1 does not work here because during Step L8 of login phase, if the adversary intercepts the login message \{CID_i, C_i, D_i, B_i\}, and blocks it, he cannot impersonate S to fool the client, since he has no knowledge of M where M is computed as h(p_i+1). The value p_i is never transmitted over the channel. Hence the proposed scheme can withstand reflection attack.

**Security against Replay Attack**

The proposed scheme is secure against replay attack, because if the adversary intercepts the messages exchanged between server and client respectively in steps L8 and A4, and tries to replay it at later time, then the receiving entity checks for the freshness of that message and since the random value in the message was already received earlier, the entity rejects the replayed login request, thus providing resistance against replay attack. Moreover, the use of nonce overcomes the
overhead of setting up the NTP and clock synchronization between client and server.

*Password Independence & Time Concurrency Weakness*

The weakness discussed in section 2.3.1 has been addressed. The computation of $\text{CID}_i, Z_i$ is completely different which rules out the possibility of password independence meaning that a wrong password entered by adversary will not lead to successful authentication.

*Loss of Smart Card*

If the user loses the smart card and if it eventually goes into the hands of the attacker, he cannot retrieve the Server’s secret key ‘x’ from $h(x \oplus h(\text{ID}_i))$ because the message digest of the user’s identity is XORed with ‘x’ and $N_i$ is formed by taking the message digest of the resultant value. The message digest $h(x \oplus h(\text{ID}_i))$ is irreversible and hence will not reveal the server’s master secret.

*Security against Stolen Verifier Attack*

Since the proposed scheme does not require a verifier at the server to authenticate a user, it is clear that the scheme is free from stolen verifier attack.

*Security against Denial of Service Attack*

Suppose, if the system administrator who has control over the server, updates any secret information of the user stored in the database
server, then the user cannot be able to login even with his valid credentials. This is called as denial of service attack. In proposed scheme, since there is no secret information stored on the server, the denial of service attack will fail.

### 3.3 FUNCTIONALITY ANALYSIS

This section presents the comparative analysis of the functionality of proposed scheme with Liao et al. scheme and shows that proposed scheme provides better security features than Liao et al. scheme.

**Table 3.1 Comparison of Functionality Analysis**

<table>
<thead>
<tr>
<th></th>
<th>Proposed</th>
<th>Liao et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Verification Table</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>User can Freely Choose and Change Password</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Security against Reflection Attack</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Security against Impersonation Attack</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Password Independence weakness</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time Concurrency problem</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Achieves Mutual Authentication successfully</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Password Change Phase</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Session Key Generation</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

### 3.4 EFFICIENCY ANALYSIS

This section discusses the comparative analysis of the efficiency of proposed scheme with Liao et al. scheme in terms of the computational cost, communication cost and the memory required in smart card & server for each user. In both the schemes hash functions, XOR operations are common operations, where as additional functions such
as random number generations and DHKE is used for computations in the proposed scheme. The detailed comparison is given in Table 3.2 below.

**Table 3.2** Comparison of Computational Efficiency

<table>
<thead>
<tr>
<th></th>
<th>Registration Phase</th>
<th>Login, Auth. &amp; Key Agreement phase</th>
<th>Client</th>
<th>Login, Auth. &amp; Key Agreement phase</th>
<th>Server</th>
<th>Password Change Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E1 (T_h)</td>
<td>E2 (T_r)</td>
<td>E3 (T_r)</td>
<td>E4 (Tx)</td>
<td>E5 (Bits)</td>
<td>E6 (Bits)</td>
</tr>
<tr>
<td>Registration Phase</td>
<td>2</td>
<td>0</td>
<td>-NA-</td>
<td>0</td>
<td>384</td>
<td>256</td>
</tr>
<tr>
<td>Liao et al.</td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
<td>256</td>
<td>0</td>
</tr>
<tr>
<td>Login, Auth. &amp; Key</td>
<td>Proposed</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>NA</td>
<td>512</td>
</tr>
<tr>
<td>Agreement phase</td>
<td>Liao et al.*</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Password Change Phase</td>
<td>Proposed</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Nil</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>Liao et al.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Registration Phase</td>
<td>Proposed</td>
<td>2</td>
<td>-NA-</td>
<td>3</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Liao et al.</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Login, Auth. &amp; Key</td>
<td>Proposed</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>-NA-</td>
<td>256</td>
</tr>
<tr>
<td>Agreement phase</td>
<td>Liao et al.</td>
<td>4</td>
<td>0</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password Change Phase</td>
<td>Proposed</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>Nil</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>Liao et al.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

* Liao et al. Scheme does not provide session key agreement

E1: Number of hash computations (T_h)
E2: Number of DHKE computations (T_r)
E3: Number of Random number generations (T_r)
E4: Number of XOR and concatenation operations (T_x)
E5: Memory needed to store user credentials;
E6: Communication cost in terms of transmitting messages and images
Here it is assumed that all the computation results are 128-bit long and the images are of 20 Kb each.

**Client Side Computations: Registration phase:** The proposed scheme performs two hash functions to compute \( h(ID_i) \) and \( h(PW_i) \) where as Liao et al. scheme performs one hash function to compute \( h(Pw_i) \). The proposed scheme computes \( h(ID_i) \) to protect user’s identity from being intercepted during transmission.

The memory required to store \( N_i, y \) and \( Q_i \) (the user’s credentials) in smart card in the proposed scheme is 384 bits, whereas in Liao et al. scheme it is 256 bits for storing \( N_i \) and \( y \). The communication cost for transmitting \( ID_i \) and \( PW_i \) from client to server in both the schemes is same i.e. 256 bits which is very less in view of the today’s high speed networks.

**Login, Authentication & key agreement phase:** The proposed scheme performs eleven hash computations, nine XOR operations, two random number generations and two DHKE computations (these numbers include the computation of authenticated session key); whereas Liao et al. scheme performs 5 hashes, ten XOR operations without session key).

The communication cost for transmitting a login message \( CID_i, C_i, D_i, B_1 \) and \( CID_i, N_i, C_i, T \) for proposed and Liao schemes respectively is 512 bits.
The Password Change phase: The computations in this phase are almost same in both the schemes except that the proposed scheme communicates with server to get the password change page. For this, the proposed scheme computes a random function and encrypts it using session key generated after successful authentication.

Server Side Computations: Registration phase: The proposed scheme performs two hash functions and three XOR / concatenation operations, whereas Liao scheme require two hash computations and one XOR operation.

The memory required to store \( h(ID_i) \) in the user profile in the proposed scheme is 128 bits whereas their scheme does not store ID_i’s on the server. Moreover, since the server sends no messages, the transmission cost is nil in both the scheme. At the end of registration, the user has to collect his smart card personally by appearing at the location.

Login, Authentication & key agreement phase, the proposed scheme performs ten hash computations, nine XOR operations, one random number generations and one DHKE computation for login, authentication and key agreement phases, whereas Liao et al.’s scheme performs four hashes and nine XOR operations. The proposed scheme utilized the DHKE to securely pass the random number between client and server to resist replay attack; whereas their scheme employs timestamp to deal with replay attack. Thus, from the above comparison it
is clear that the proposed scheme requires six extra hashes and one DHKE computation comparing to their scheme, but to achieve complete security against various online attacks, these extra computations would not add overhead on server. The transmission cost for both the schemes is 256 bits i.e. $A_1$ and $M_1$ in case of proposed scheme and $M_1, T$ in Liao scheme.

*The Password Change phase:* The computations in this phase are almost same in both the schemes at client except that the proposed scheme requires password change page from the server. For this, proposed scheme computes a random function and encrypts it using session key generated after successful authentication.

### 3.5 FORMAL VERIFICATION

This section presents the verification result and source code of the scheme written in Security Protocol Description Language (.spdl), default language of Scyther. As discussed in the section 2.3.3, the agents perform the roles and each role defines the sequence of actions that are performed in a role.

As per the specification the Agent model in the proposed scheme has two agents i.e. ‘C’ – Client and ‘S’ – Server. Each agent performs the roles, hence, there are two roles that are named after the agents i.e. ‘role C’ and ‘role S’. Role Event in the scheme is role H which is defined to handle the intermediate computations. The adversary model is also
designed considering that the adversary has complete control of the network. Since Scyther checks for the freshness and synchronization by default, the claims include the above discussed features as claim events.

usertype SessionKey;
secret k: Function;
const succ: Function;
const Fresh: Function;
const Compromised: Function;
const hash: Function;
const XOR: Function;
const plus: Function;
const compare: Function;

protocol StarCompromise(H)
{
  // Read the names of 3 agents and disclose a session between them including corresponding session key to simulate key compromise
  role H {
    const a1,a2, A1,A2,Skey,
b1,b2,B1,B2,ID,Qi,Ri,Pi,Pj,Vi,Pwi,Images,1,Y: Nonce;
    var S,C: Agent;
    read_!H1(H,H, S,C);
    send_!H2(H,H, {a1}A1, {b1}B1, {{B1}Pi}k(C,S),
        XOR(hash(ID),hash(Pi)), XOR(hash(Pwi),hash(XOR(Pi,Y,
        XOR(hash(Pwi),hash(XOR(Skey,hash(ID)))))));
    }

    # claim_H3(C,Empty, (Compromised));
  }
}

protocol JDIM(S,C)
{
  role S {
    const a1,a2, A1,A2,Skey,
b1,b2,B1,B2,ID,Qi,Ri,Pi,Pj,Vi,Pwi,Images,1,Y: Nonce;
    send_1(S,C, {a1}A1 );
    read_2(C,S, {b1}B1, {{B1}Pi}k(C,S), XOR(hash(ID),hash(Pi)),
        XOR(Pwi),hash(XOR(Skey,hash(ID)))));
    }
}
XOR(hash(Pwi),hash(XOR(Skey,hash(ID)))))
     hash(XOR(Pi,Y,hash(XOR(Pwi),hash(XOR(hash(Pwi),hash(XOR(hash(Pwi),hash(XOR(Pi,Y,
     XOR(hash(Pwi),hash(XOR(Skey,hash(ID))))))))))))))
     );
     send_3(S,C, hash(plus(Pi,1)));
     claim_S1(S,Nisynch);
     claim_S2(S,Nagree);
     claim_S3(S,Secret,hash(plus(Pi,1)));
     claim_S4(S,Secret,XOR(hash(ID),hash(Pi)));
     claim_S5(S,Secret,XOR(hash(Pwi),hash(XOR(Pi,Y,XOR(hash(Pwi),hash(XOR(Skey,hash(ID)))))))));
     claim_S6(S,Secret,hash(XOR(Pi,Y,hash(XOR(Pwi),hash(XOR(hash(Pwi),hash(XOR(hash(Pwi),hash(XOR(Pi,Y,
     XOR(hash(Pwi),hash(XOR(Skey,hash(ID)))))))))))))))));
     claim_S7(S,Secret,hash(plus(Pi,1)));
   }
}

role C{
  const a1,a2, A1,A2, Skey,
  b1,b2,B1,B2,ID,Qi,Ri,Pi,Pj,Vi,Pwi,Images,1,Y: Nonce;
  read_1(S,C, {a1}A1);
  send_2(C,S, {b1}B1, {{B1}Pi}{k(C,S), XOR(hash(ID),hash(Pi)),
     XOR(hash(Pwi),hash(XOR(hash(Pwi),hash(XOR(hash(Pwi),hash(XOR(Skey,hash(ID))))))))},
     hash(XOR(Pi,Y,hash(XOR(hash(Pwi),hash(XOR(Pw
     i),hash(XOR(hash(Pwi),hash(XOR(hash(Pwi),hash(XOR(Skey,hash(ID))))))))))))));
  read_3(S,C, hash(plus(Pi,1)));
  claim_C1(C,Nisynch);
  claim_C2(C,Nagree);
  claim_C3(C,Secret,hash(plus(Pi,1)));
  claim_C4(C,Secret,XOR(hash(ID),hash(Pi)));
  claim_C5(C,Secret,XOR(hash(Pwi),hash(XOR(hash(Pwi),hash(XOR(hash(Pwi),hash(XOR(hash(Pw
     i),hash(XOR(hash(Pwi),hash(XOR(hash(Pwi),hash(XOR(Skey,hash(ID))))))))))))));
  claim_C6(C,Secret,hash(XOR(Pi,Y,hash(XOR(hash(Pwi),hash(XOR(hash(Pwi),hash(XOR(hash(Pw
     i),hash(XOR(hash(Pwi),hash(XOR(Skey,hash(ID))))))))))))));
  claim_C7(C,Secret,hash(plus(Pi,1)));
}

const Alice,Bob,Eve: Agent;

untrusted Eve;
const ne: Nonce;
const kee: SessionKey;
compromised k(Eve,Eve);
compromised k(Eve,Alice);
compromised k(Eve, Bob);
compromised k(Alice, Eve);
compromised k(Bob, Eve);

Figure 3.4 Screenshot of Verification Result