Project:
Implementation of authentication scheme using PBC (Pairing Based Cryptography) library C or JAVA (JPBC)

Group Number - 3

- Manish K. Raushan (2021801005)
- Sudipta Halder (2021202011)
- Nitin Kumar (2021202020)

Introduction

- The implementation of the authentication scheme is based on the information security scheme proposed in the paper "**Efficient privacy preserving device authentication in WBANs for industrial e-health applications**".

- Proposed scheme provides robust security and avoids the management of large number of public-keys of application providers by the client device.

- Implementation using the PBC (Pairing-Based Cryptography), a C library that performs the mathematical operations underlying pairing-based cryptosystems

- PBC provides cryptographic functionalities such as elliptic curve generation, elliptic curve arithmetic and pairing computation.

Authentication scheme

1. The scheme consists of three entities known as
   I. **Network Manager NM**
   II. **Application Provider AP**
   III. **Client C**
2. NM sets up its parameters during the initialization phase.
3. Client and AP register with NM
4. AP and Client receive their corresponding authentication credentials from NM
5. Perform the authentication by exchanging the messages
6. After successful authentication, establish a secure session between them for secure communication

Authentication Parameters

- G_1 , G_2 are Bilinear groups
- P *is* Generator of G_1
- g *is* Generator of G_2 ; $g = e\,(P, P)$
- q *is* Prime order of G_1 and G_2
- h_1 , h_2 are Secure hash functions, where $h_1 , h_2 : \{\,0 , 1\,\}^* \longrightarrow Z^*_q$
- (s_{nm} , Q_{nm}) is Private and public key pairs of NM; $Q_{nm} = s_{nm}\,P$
- AP, ID_{ap} is Application provider, and its identity
- (s_{ap} , K_{ap}) is Private key pair of AP ; $s_{ap} = h_1\,(K_{ap} \,||\, ID_{ap}\,)$
- C, ID_c are client, and its identity
- (s_c, g_c) is Key pair of C , where s_c is kept secret
- $x \leftarrow_R S$ x is randomly picked from a set S

Phases of authentication process

Initialization phase

1. NM chooses bilinear pairing groups $\{ G_1 , G_2 \}$ of order q , with generators $P \in G_1$ and $g = e\,(P, P) \in G_2$, where e is a bilinear pairing operation defined as $e : G_1 \times G_1 \longrightarrow G_2$.
2. It also chooses the symmetric-key cryptography (In our implementation, we consider *AES*) and two cryptographic one-way hash functions $h_1 : \{ 0 , 1 \}^* \longrightarrow Z^*_q$ and $h_2 : \{ 0 , 1 \}^* \longrightarrow Z^*_q$.
3. NM then generates its master private key $s_{nm} \xleftarrow{R} Z^*_q$ and computes public keys $Q_{nm} = s_{nm} P \in G_1$ and $g_{nm} = g\, s_{nm} \in G_2$.
4. Finally, NM declares the public parameters $\{ G_1 , G_2 , q, e, P, Q_{nm} , g_{nm}$ $g, h_1 , h_2 , \}$.

Phases of authentication process

Client Registration phase

1. C sends its chosen unique identity ID_c to NM via a secure channel.
2. Upon receiving a request from C, NM checks its validity and then defines its access right as $Right_c = [\, EID_c \,||\, right \,||\, Lifetime \,]$, where $EID_c = E_{s\,nm}(ID_c \,||\, Lifetime)$.
3. Next, NM chooses $r_c \leftarrow R - Z^*_q$ and computes $g_c = g^{r_c}$ and with the condition $s_c = r_c + s_{nm} h_c$, where $h_c = h_1(g_c, Right_c, Q_{nm})$
4. NM sends $\{ Right_c, s_c, g_c \}$ to C via a secure channel.
5. C keeps secret the received credentials $\{ Right_c, s_c, g_c \}$.

Phases of authentication process

AP Registration phase

1. AP sends its chosen identity ID_{ap} to NM via a secure channel.
2. After receiving a request from AP, NM checks its validity and then computes $K_{ap} = [1/\{h_1(ID_{ap}) + s_{nm}\}]\, P$.
3. Finally, NM sends K_{ap} to AP via a secure channel.
4. Upon receiving credential K_{ap} , AP computes $s_{ap} = h_1 (K_{ap} \,||\, ID_{ap})$
5. AP keeps secret the pair (K_{ap} , s_{ap}) and
6. publicly declares its identity ID_{ap} .

Phases of authentication process

Authentication and key establishment phase

1. C chooses $x_c \xleftarrow{R} Z_q^*$ and publicly available identity ID_{ap} of AP.
2. Then, C computes $T_1 = x_c(h_1(ID_{ap})P + Q_{nm})$, $k_1 = g\, x_c$, $C_1 = E\, k_1[g_c, Right_c, t_1]$, and $Auth_1 = h_2(T_1, g_c, Right_c, t_1, k_1)$, where t_1 is the current time stamp.
3. *C* sends $m_1 = \{T_1, C_1, Auth_1\}$ to AP.
4. After receiving m_1, AP computes $k_2 = e(T_1, K_{ap})$, and retrieves $[g_c, Right_c, t_1]$ by decrypting C_1 using computed k_2.
5. AP then checks the validity of t_1 and $Right_c$. If these are valid, AP further checks whether $Auth_1 = h_2(T_1, g_c, Right_c, t_1, k_2)$ holds or not.

6. If the received message m_1 is valid, AP next generates $y_{ap} \xleftarrow{R} Z_q^*$ and computes

$$
\begin{aligned}
h_c &= h_1(g_c, Right_c, Q_{nm}) \\
y_1 &= g_c \times g_{nm}^{h_c} = g^{s_c} \\
y_2 &= y_1^{y_{ap}+s_{ap}} = g^{s_c(y_{ap}+s_{ap})} \\
sk_{ap} &= (y_1 \times k_2)^{y_{ap}+s_{ap}} \\
Auth_2 &= h_2(y_2, sk_{ap}, g_c, ID_c, T_1, k_2, t_2)
\end{aligned}
$$

   here t_2 is the current timestamp.
7. Finally, AP sends the challenge message $m_2 = \{y_2, Auth_2, t_2\}$ to *C*.

Phases of authentication process

Authentication and key establishment phase Contd…

8. Upon receiving m_2 , C checks the validity of t_2 . Then C computes $sk_c = y_2 (x_c + s_c) / s_c$, and verifies the validity of the condition $Auth_2 = h_2 (y_2 , sk_c , g_c , ID_c , T_1 , k_1 , t_2)$. If it is valid, C authenticates AP and confirms that the shared session key is sk_c.
9. Finally, C computes the confirmation message $Auth_3 = h_2 (sk_c , k_1 , y_2 , t_1 , t_2)$, and sends $m_3 = \{ Auth_3 \}$ to AP.
10. After receiving m_3 , AP checks the validity of the condition $Auth_3 = h_2 (sk_{ap} , k_2 , y_2 , t_1 , t_2)$. If it is valid, AP confirms that the C is legitimate and agrees on the session key sk_{ap} .

Implementation & Testing

- **The following parameters are initialized by Network Manager:**
  - Generator P ∈ G1
  - Generator g = e (P, P) ∈ G 2
  - Network Manager’s private key is generated S_{nm} <- Z_q*
  - Network Manager’s public keys are generated i) Q_{nm} <- S_{nm}P ∈ G1, ii) g_{nm} <- g^{Snm} ∈ G2
  - Following are the code snippet and corresponding output

```c
/********NM Parameter generation********************/
    element_init_G1(P, pairing);
    element_random(P);
    element_printf("system parameter P = %B\n", P);

    element_init_GT(g, pairing);
    element_pairing(g, P, P);
    element_printf("system parameter g = %B\n", g);

    element_init_Zr(Snm, pairing);
    element_random(Snm);
    element_printf("system parameter Snm = %B\n", Snm);

    element_init_G1(Qnm, pairing);
    element_mul_zn(Qnm,P,Snm);
    element_printf("system parameter  Qnm= %B\n", Qnm);

    element_init_GT(Gnm, pairing);
    element_pow_zn(Gnm, g, Snm);
    element_printf("system parameter Gnm = %B\n", Gnm);
```

```
system parameter P = [830897818227176345408587560734596872266385099775296117424544298536822918519281194099890453738695467214035469527723191409523290087385167977550880 47
18991351, 879382875826047809491720503611060878656491343344253183155645481819201899767542750941319761145557080685782405321767689741157232811990241941541084363871612]

system parameter g = [72508836051430531491690373567604068537773631098195947621114189788875132222914277411336930707735774951052528794442703168093372955274456186896635977
39829784, 4912167151152519963450928837333901511225835558549003081270380838508073642591675465911879293768498774128634647694335174652688347650720013087102729658602498]

system parameter Snm = 3181350158940477909537860774289350138616751094

system parameter  Qnm= [7676357200007757458373121482442289572732381180244758788974330294696624144905170546771023288820925101450378325870847567797199440981057733352755 80
7007470699, 7253584587860556538319291839474500130464393744517391883278314312631460982564758262817846386217720582397312033549587246697516568504027089613788863634587995]

system parameter Gnm = [802772955963978641896906021413000499610877880423831881010804200381084045992072064213317816366715224639660266999151885596712421935758404122236419
7805801996, 5913197798958153059459121734429217368268265501532046709890577847621366118840283366516203032842223913284846052268164024133392841679657691049686744018829148]
```

Implementation & Testing

- **Client Registration Phase**
  - Client Choose ID_c and send it to Network Manager.
  - Network Manager checks validity of ID_c
  - Network manager computes $r_c \leftarrow Z_q^*$
  - Network Manager also computes $g_c = g^{rc}$, $s_c = r_c + s_{nm} h_c$ ($h_c = hash(g_c, Right_c, Q_{nm})$)
  - Network Manager sends these parameters to Client and Client stores them ($Right_c$, s_c, g_c) for future reference.
  - Picture shown are code snippet and corresponding output.

```c
/*******************Additional NM Parameter******************/
    element_init_Zr(Rc, pairing);
    element_random(Rc);
    element_printf("system parameter Rc = %B\n\n", Rc);

    element_init_GT(Gc, pairing);
    element_pow_zn(Gc, g, Rc);
    element_printf("system parameter Gc = %B\n\n", Gc);

    element_init_Zr(Hc, pairing);
    element_from_hash(Hc, "gc, Rightc, Qnm", 16);
    element_printf("system parameter Hc = %B\n\n", Hc);

    element_init_Zr(SnmHc, pairing);
    element_mul_zn(SnmHc,Snm,Hc);
    element_printf("system parameter SnmHc= %B\n\n", SnmHc);

    element_init_Zr(Sc, pairing);
    element_add(Sc,Rc,SnmHc);
    element_printf("system parameter Sc= %B\n\n", Sc);
```

```
system parameter Rc = 730716962802279865662429872872194594894088373312

system parameter Gc = [4106207343734189224851445340325883624254746756538529109132278805673329146814872591223847241298620556639017061460949020313251548739358711132377247
439562513, 4879700238257413257395970472492073908533737797058104103420140760251825241563683521321031488224699746866766807022640774707997302731480128177685994154250086]

system parameter Hc = 590237667108670725796356627062109070680260043564

system parameter SnmHc= 193520636605864273285334413686017136957387823977

system parameter Sc= 193486780742692517586645040986706830445499637672
```

Implementation & Testing

- Image.1 shows how Network manager transfers the parameters to Client and Application Provider over network.
- Image 2. shows how Client and AP gets the parameters over network from NM.

```
Size of P is 128
Size of Snm is 20
Size of Qnm is 128
Size of g is 128
Size of Gnm is 128
Size of Sc is 20
Size of Gc is 128
```

```c
element_to_bytes(P_bytes,P);
element_to_bytes(Snm_bytes,Snm);
element_to_bytes(Qnm_bytes,Qnm);
element_to_bytes(g_bytes,g);
element_to_bytes(Gnm_bytes,Gnm);
element_to_bytes(Sc_bytes,Sc);
element_to_bytes(Gc_bytes,Gc);


printf("\nSize of P is %d",element_length_in_bytes(P));
printf("\nSize of Snm is %d",element_length_in_bytes(Snm));
printf("\nSize of Qnm is %d",element_length_in_bytes(Qnm));
printf("\nSize of g is %d",element_length_in_bytes(g));
printf("\nSize of Gnm is %d",element_length_in_bytes(Gnm));
printf("\nSize of Sc is %d",element_length_in_bytes(Sc));
printf("\nSize of Gc is %d\n",element_length_in_bytes(Gc));

memcpy(buf,P_bytes,128);
memcpy(buf+128,Snm_bytes,20);
memcpy(buf+148,Qnm_bytes,128);
memcpy(buf+276,g_bytes,128);
memcpy(buf+404,Gnm_bytes,128);
memcpy(buf+532,Sc_bytes,20);
memcpy(buf+552,Gc_bytes,128);


            if (sendto(sfd, buf, 680 , 0, (struct sockaddr*) &client_address, sizeof(client_address)) == -1)
{
  printf("Sendto failed\n");
}
            if (sendto(sfd, buf, 680 , 0, (struct sockaddr*) &ap_address, sizeof(ap_address)) == -1)
{
  printf("Sendto failed\n");
```

Image -1

```
Waiting for message from NM...
Received Authentication message from NM of size 680
P is [830897818227176345408587560734596872266385099775296117424544298536822918519281194099890453738695467214035469527723191409523290087385167977550880471899135 1, 8793828758260478094917205036110608786564913433442531831556454818192018997675427509413197611455570806857824053217676897411572328119902419415410843638 71612]

Snm is 318135015894047790953786077428935013861675 1094

Qnm is [767635720000775745837312148244228957273238118024475878897433029469662414490517054677102328882092510145037832587084756779719944098105773335275580700747 0699, 72535845878605565383192918394745001304643937445173918832783143126314609825647582628178463862177205823973120335495872466975165685040270896137888636345 87995]

g is [725088360514305314916903735676040685377736310981959476211141897888751322229142774113369307077357749510525287944427031680933729552744561868966359773982 9784, 49121671511525199634509288373339015112258355585490030812703808385080736425916754659118792937684987741286346476943351746526883476507200130871027296586 02498]

Gnm is [802772955963978641896906021413000499610877880423831881010804200381084045992072064213317816366715224639660266999151885596712421935758404122236419780580 1996, 59
```

```
Waiting for message from NM...
Received Authentication message from NM of size 680

P is [8308978182271763454085875607345968722663850997752961174245442985368229185192811940998904537386954672140354695277231914095232900873851679775508804718991351, 879382875826047809491720503611060878656491343344253183155645481819201899767542750941319761145557080685782405321767689741157232811990241941541084363871612]

Snm is 3181350158940477909537860774289350138616751094

Qnm is [7676357200007757458373121482442289572732381180244758788974330294696624144905170546771023288820925101450378325870847567797199440981057733352755807007470699, 7253584587860556538319291839474500130464393744517391883278314312631460982564758262817846386217720582397312033549587246697516568504027089613788863634587995]

g is [7250883605143053149169037356760406853777363109819594762111418978887513222291427741133693070773577495105252879444270316809337295527445618689663597739829784, 4912167151152519963450928837333901511225835558549003081270380838508073642591675465911879293768498774128634647694335174652688347650720013087102729658602498]

Gnm is [80277295596397864189690602141300049961087788042383188101080420038108404599 20
```

Image -2

Implementation & Testing

- **Application Provider(AP) Registration Phase**
  - AP chooses one ID_{ap}
  - AP sends ID_{ap} to NM over network..
  - NM checks the validity of ID_{ap} sent by AP.
  - NM calculates $K_{ap} = P/(hash(ID_{ap}) + S_{nm})$
  - NM sends this K_{ap} to AP over network..
  - AP calculates $S_{ap} = hash(K_{ap} || ID_{ap})$
  - AP then makes ID_{ap} public and stores the pair (K_{ap}, S_{ap}).
  - Refer to the below pictures for code snippet and corresponding output.

```
element_init_Zr(h1, pairing);
element_from_hash(h1, "ID(AP)", 7);
element_printf("system parameter h1 = %B\n\n", h1);

element_init_Zr(h1plusSnm, pairing);
element_add(h1plusSnm,h1,Snm);
element_printf("system parameter h1plusSnm = %B\n\n", h1plusSnm);

element_init_Zr(Invh1plusSnm, pairing);
element_invert(Invh1plusSnm,h1plusSnm);
element_printf("system parameter Invh1plusSnm = %B\n\n", Invh1plusSnm);

element_init_G1(Kap, pairing);
element_mul_zn(Kap,P,Invh1plusSnm);
element_printf("system parameter Kap = %B\n\n", Kap);
```

```
element_init_Zr(Sap, pairing);
element_from_hash(Sap, "Kap||ID(AP)", 12);
element_printf("system parameter Sap = %B\n", Sap);
```

```
system parameter h1 = 418276283659982611770177922534362380329835309121

system parameter h1plusSnm = 421457633818923089679715783308651730468452060215

system parameter Invh1plusSnm = 716796731548057768253613340415198032419459840160

system parameter Kap = [3649535029595507351207834449369123985691233468426106640650603600681973719587608598791913530609177918800904448910387219147773274216455629441174898172859962, 59436041568036537700824111765965214817646688973752217116806567111814901142789110962482233136765761702437950768350776314717832913779136684728622639750728
30]
```

Implementation & Testing

- **Authentication Phase between Client and AP**
  - **Client C Authentication message(m_1) generation:**
    - Client computes x_c <- Z_q*
    - It also calculates $T_1 = x_c(hash(Id_{ap})P + Q_{nm})$.
    - It also calculates $k_1 = g^{xc}$, $C_1 = Enc(g_c, Rights_c\ t_1)$
    - Then it calculates $Auth_1 = hash(T_1, g_c, Right_c, t_1, k_1)$
    - Then it sends message $m_1 = (T_1, C_1, Auth_1)$ to AP over network..
    - We also have calculated time required for performing hash operation and encryption operation and also communication delay over network.
    - For hashing we have used **SHA-256**, and for **encryption** we have used **AES.**

```c
element_to_bytes(T1_bytes,T1);
memcpy(buf,T1_bytes,element_length_in_bytes(T1));

element_to_bytes(K1_bytes,K1);
sha256(K1_bytes,128,hash);
memcpy(enc_key,hash,32);
//printf("Sizeof element_t Gc is %d\n",element_length_in_bytes(Gc));
element_printf("\nGc is %B",Gc);
element_to_bytes(Gc_bytes,Gc);
gettimeofday(&now,NULL);
prev_time=now.tv_sec*1000000+now.tv_usec;
encrypt_aes(Gc_bytes,128,enc_buf,enc_key);
gettimeofday(&now,NULL);
pres_time=now.tv_sec*1000000+now.tv_usec;
printf("\nTime for encryption (AES-256) computation time is %d microseconds\n",(pres_time-prev_time));


memcpy(buf+128,enc_buf,element_length_in_bytes(Gc));


memcpy(T1Gc,T1_bytes,128);
memcpy(T1Gc+128,Gc_bytes,128);
memcpy(T1Gc+256,K1_bytes,128);
gettimeofday(&now,NULL);
prev_time=now.tv_sec*1000000+now.tv_usec;
sha256(T1Gc,384,hash);
memcpy(buf+256,hash,32);
gettimeofday(&now,NULL);
pres_time=now.tv_sec*1000000+now.tv_usec;
printf("Time for SHA-256 computation time is %d microseconds\n",(pres_time-prev_time));

if (sendto(sfd, buf, 288, 0, (struct sockaddr*) &ap_address, sizeof(ap_address)) == -1)
        {
            printf("Sendto failed\n");
```

```
system parameter T1 = [623510651305075509532297369663451124505773356448489684998217180913873752689547647133930718316336188789857829259983318363734854218087646738468274 1
006218465, 753857073828150889299663455223259353064099208238599488180909456369425071079441699474010356263341905836920513815964097034023096632343102412255712917569503 3]

system parameter K1 = [450221024906281041637959726528131165230626975776929620470535670587493929841945633201383622605606707825842208246157801702877833093656108798063884 3
752342484, 864466197986427429354366397560971322736910496819904638425716917919719713902336552434117999721389242633178914687410741210773290282388841311270405846224195 2]


Gc is [410620734373418922485144534032588362425474675653852910913227880567332914681487259122384724129862055663901706146094902031325154873935871113237724743956 2513, 48797
002382574132573959704724920739085337377970581041034201407602518252415636835213210314882246997468667668070226407747079973027314801281776859941542 50086]

Time for encryption (AES-256) computation time is 21 microseconds
Time for SHA-256 computation time is 6 microseconds
Waiting for message from AP...

Communication delay over network is 33 microseconds
```

Implementation & Testing

- **AP Phase-2 Authentication i.e. Message m2 Generation**
  - AP calculates $k_2 = Enc(T_1, K_{ap})$
  - AP decrypts C_1 received from Client ($Dec_{k2}(C_1)$) and retrieves [g_c\`, $Right_c$\`, t_1\`].
  - Then it confirms the validity of t_1\` and $Right_c$\`. If they match, then accept or Reject.
  - Now, AP calculates hash(T_1, g_c\`, $Right_c$\`, t_1\`, k_2). If it matches with $Auth_1$ which has been sent by Client, then accept, otherwise reject.
  - If the above step is successful, then AP generates $y_{ap} <- Z_q^*$.
  - AP calculates $h_c = hash(g_c, Right_c, Q_{nm})$.
  - Then, AP calculates $y_2 = y_1^{(yap + sap)}$
  - Then, AP calculates $sk_{ap} = (y_1 * k_2)^{(yap + sap)}$
  - Then AP calculates $Auth_2 = hash(y_2, sk_{ap}, g_c, ID_c, T_1, k_2, t_2)$.
  - Now, AP constructs a message $m_2 = (y_2, Auth_2, t_2)$ and sends it to Client over network.
  - Refer to the below pictures for code snippet and corresponding output.

```
/*************Phase-2 Authentication i.e. Message m2 Generation******************/
element_init_Zr(Yap, pairing);
element_random(Yap);
element_printf("system parameter Yap = %B\n", Yap);

element_init_Zr(Sap, pairing);
element_from_hash(Sap, "Kap||ID(AP)", 12);
element_printf("system parameter Sap = %B\n", Sap);

element_init_GT(Y1, pairing);
element_pow_zn(Y1, g, Sc);
element_printf("system parameter Y1 = %B\n", Y1);

element_init_Zr(YapPlusSap, pairing);
element_add(YapPlusSap,Yap,Sap);
element_printf("system parameter YapPlusSap = %B\n", YapPlusSap);

element_init_GT(Y2, pairing);
element_pow_zn(Y2, Y1,YapPlusSap);
element_printf("system parameter Y2 = %B\n", Y2);

element_init_GT(Y1K2, pairing);
element_mul(Y1K2, Y1,K2);
element_printf("system parameter Y1K2 = %B\n", Y1K2);

element_init_GT(SKap, pairing);
element_pow_zn(SKap, Y1K2,YapPlusSap);
element_printf("system parameter SKap = %B\n", SKap);

element_to_bytes(Y2_bytes,Y2);
element_to_bytes(SKap_bytes,SKap);

memcpy(buf,Y2_bytes,128);
```

Implementation & Testing

AP Phase-2 Authentication i.e. Message m2 Generation output window is shown below

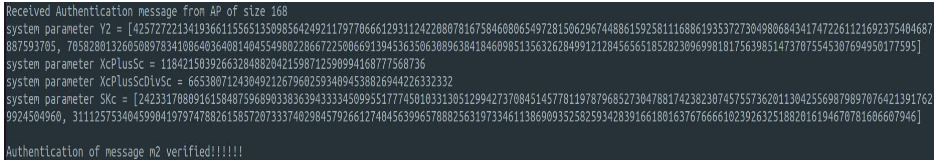
```c
memcpy(Y2SKap,Y2_bytes,128);
memcpy(Y2SKap+128,SKap_bytes,128);
memcpy(Y2SKap+256,Gc_bytes,128);
memcpy(Y2SKap+384,T1_bytes,128);
memcpy(Y2SKap+512,K2_bytes,128);




/*printf("\nY2SKap is");
for(int i=0;i<640;i++)
{
  if(i%128==0)
  printf("\n");
  printf("%hhu ",Y2SKap[i]);
}*/


sha256(Y2SKap,640,hash);

memcpy(buf+128,hash,32);
gettimeofday(&now,NULL);
time2=now.tv_sec*1000000+now.tv_usec;
//printf("\ntime2 is %lu",time2);
memcpy(buf+160,(unsigned char*)&time2,8);



if (sendto(sfd, buf, 168 , 0, (struct sockaddr*) &client_address, sizeof(client_address)) == -1)
{
  printf("Sendto failed\n");
}
```

```
system parameter K2 = [45022102490628104163795972652813116523062697577692962047053567058749392984194563320138362260560670782584220824615780170287783309365610879806388
43752342484, 86446619798642742935436639756097132273691049681990463842571691791971971390233655243411799972138924263317891468741074121077329028238884131127040584622419522]


system parameter Gc = [41062073437341892248514453403258836242547467565385291091322788056733291468148725912238472412986205566390170614609490203132515487393587111323772
47439562513, 48797002382574132573959704724920739085337377970581041034201407602518252415636835213210314882246997468667668070226407747079973027314801281776859941542500866]

Authentication of message m1 verified!!!!!!
system parameter Yap = 720644737266238686591989891996708074700915811316
system parameter Sap = 430347279032287981647404239704611786255509506372
system parameter Y1 = [30732900629572154603238917714388281867468814004875500329619055925323913312496129756169012011510181774678153757209740515876913367172185796222239
50939971344, 20937941709209679799567309766072058266011414716394674450589485277889282244472157233936974604973374147869735546848650215881963448466516290444065649886313322]
```

```
system parameter YapPlusSap = 420241197633075046878274886129814959550448758071
system parameter Y2 = [4257272213419366115565135098564249211797706661293112422080781675846080654972815062967448861592581116886193537273049806843417472261121692375404687887593705, 70582801326050897834108640364081404554980228667225006691394536350630896384184609851356326284991212845656518528230969981817563985147370755453076949501775955]
system parameter Y1K2 = [4881329116647759760395812730778228751450348617708679602524234258397929242976772652118839550623891614082038125984485315439625141471649739621295734977866433, 5216498950359925981564957950421898937137616300770399006368934845629770789621252183563466465890207834783957722572729934973516102515471981354124795773614320]
system parameter SKap = [242331708091615848759689033836394333345099551777450103313051299427370845145778119787968527304788174238230745755736201130425569879897076421391
7629924504960, 311125753404599041979747882615857207333740298457926612740456399657888256319733461138690935258259342839166180163767666610239263251882016194670781606607946]
```

Implementation & Testing

- **Client Phase-3 Authentication i.e. Message m3 Generation**
  - Client checks the validity of t_2 sent by AP.
  - Client calculates $sk_c = y_2^{(xc+sc)/sc}$
  - Then, Client calculates hash(y_2, s_k, g_c, ID_c, T_1, k_1, t_2), if it matches with $Auth_2$ then Client accepts message m_2 otherwise rejects it.
  - If the above step is true, then Client calculates Auth3 = hash(sk_c, k_1, y_2, t_1, t_2) and sends it as $m_3 = Auth_3$ to AP.
  - Refer to the below pictures for code snippet and corresponding output.



Implementation & Testing

- **AP Phase-4 Authentication**
  - AP checks $Auth_3$ = hash(sk_{ap}, k_2, y_2, t_1, t_2) or not. If equal, accept, otherwise reject.
  - Refer to the below pictures for code snippet and corresponding output.

```c
printf("Waiting for message from client...\n");
len=recvfrom(sfd,buf,1024,0,0,0);
printf("Received Authentication message from Client of size %d\n",len);

memcpy(SKapK2,SKap_bytes,128);
memcpy(SKapK2+128,K2_bytes,128);
memcpy(SKapK2+256,Y2_bytes,128);
sha256(SKapK2,384,hash);

if(memcmp(hash,buf,32)==0)
{
    printf("\nAuthentication of message m3 verified!!!!!!\n");
    printf("***************Authentication Successful*******************\n");
}
else
{
    printf("\nAuthentication of message m3 failed????????\n");
    printf("\nXXXXXXXXXXXXXXXXXAuthentication FailedXXXXXXXXXXXXXXXXXXXXXX\n");
}
```

```
Waiting for message from client...
Received Authentication message from Client of size 32

Authentication of message m3 verified!!!!!!
**************Authentication Successful*****************
```

Performance Analysis and Future Work

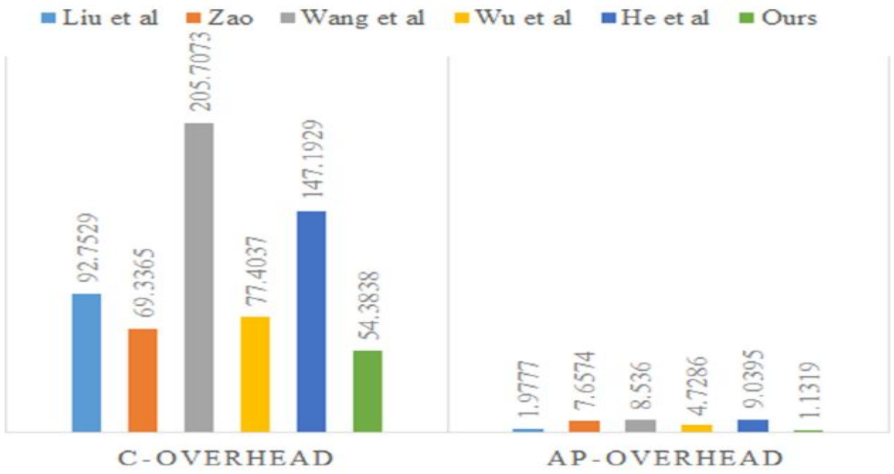


Fig. 2 – Comparison of computational cost of C and AP in milliseconds.

Scheme	Communication overhead (in bits)
Liu et al. (2014)	3424
Zhao (2014)	3648
Wang and Zhang (2015)	3268
Wu et al. (2016)	3168
He et al. (2016)	3328
Our proposed scheme	2208

Comparison of Communication overhead

Future Work:

- Encode and decode the messages using CBOR (Concise Binary Object Representation). Also it helps to compress the message size which will reduce the communication overhead.

Thank You