1+1=2 X

```java
// CS Setup Phase
System.out.println("\n--- CS Setup Phase ---");
//CS chooses 160 bit MSK and k
MSK = new BigInteger(160, rnd);
k = new BigInteger(160, rnd);

//CS chooses public system parameter
n = BigInteger.probablePrime(300, rnd);

//CS sets its identity IDs
BigInteger nsid = new BigInteger(16,rnd);
IDs = "serverID"+nsid.toString();
System.out.println("IDs = "+IDs);

//CS computes PIDs
PIDs = Hash(IDs+k.toString());

//Print the Secret and Public Key in HEX
System.out.println("Secret Key (HEX)");
System.out.println("MSK = "+MSK.toString(16));
System.out.println("k = "+k.toString(16));

System.out.println("\nPublic Key (HEX)");
System.out.println("h : using SHA-256");
System.out.println("n = "+n.toString(16));
System.out.println("PIDs = "+PIDs.toString(16));
```

(1)

(2)

$h(ID_s \| k)$

output

```
--- CS Setup Phase ---
IDs = serverID55642
Secret Key (HEX)
MSK = 8092cb2fd809c3cfba68571e43da32a09b78f995
k = f8262d0811a0c40e3094c4af24f8100cd2034692
```

```
Public Key (HEX)
h : using SHA-256
n = e68476f4257dec5d3b40b816a4bc3e6d0f4163416c08e8bb2280834e116febdeef1d9c4ff8f
PIDs = b4424e0ab4779ec7d72f6b48169a14e63176edbfeadacd9967b77e6e25c68692
```

4.1. Setup phase

In this phase, CS generates its master private key and other public system parameters in the following steps:

1. CS randomly chooses a 160 bits numbers MSK as its master private key, and then chooses a 160 bits mask key k and the public system parameter n.
2. CS chooses a secure one-way hash function $h : \{0,1\}^* \rightarrow Z_n^*$, its identity ID_s and computes $PID_s = h(ID_s \| k)$.
3. CS saves (MSK, k) secretly and publishes (h, n, PID_s).

SHA-256.

output 256 bits.

For example:

a = "program" , b = "Java"

a||b = "programJava"

concat

⊕ : XOR

Bit 1, 0

a	b	a ⊕ b
0	0	0
1	0	1
0	1	1
1	1	0

2-1

0 (1)

For example : Integer 6 (Base 2)

Binary $(110)_2 = 6$

$1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 6$

Method to compute Hash Function

```java
private static BigInteger Hash(String hash) throws NoSuchAlgorithmException, UnsupportedEncodingException {
    //This method to compute Hash from string as an input
    //could change to SHA1,SHA-128, SHA-256, SHA-512
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.update(hash.toString().getBytes("UTF-8"));
    byte[] digest = md.digest();

    StringBuffer sb = new StringBuffer();
    for (int i = 0; i < digest.length; i++) {
     sb.append(Integer.toString((digest[i] & 0xff) + 0x100, 16).substring(1));
    }

    return new BigInteger(sb.toString(),16);
```

SHA : Secure Hash Algorithm.

Decimal and HEX representation

10-1

Dec : 0, 1, 2, ..., 9 ; $279 = 2 \cdot 10^2 + 7 \cdot 10^1 + 9 \cdot 10^0$. (Base 10)

Hex : 0 (Base 16) 0 1 2 ... 9 10 11 12 13 14 15 (16-1)

A B C D E F.

```java
//Drone Registration Phase
System.out.println("\n--- Drone Registration Phase ---");
//Drone sets its identity IDj
BigInteger droneid = new BigInteger(16,rnd);
IDj = "droneID"+droneid.toString();
System.out.println("IDj = "+IDj);

//CS computes PIDj and alpha_j for Drone
PIDj = Hash(IDj+k.toString());
alphaj = Hash(IDj+MSK.toString());
System.out.println("PIDj and alpha_j from CS (HEX)");
System.out.println("PIDj = "+PIDj.toString(16));
System.out.println("alpha_j = "+alphaj.toString(16));

//CS stores (IDj, alpha_j, PIDj) in Ls
Ls[0][0] = IDj;
Ls[0][1] = alphaj.toString();
Ls[0][2] = PIDj.toString();
```

(1)

(2)

16

Ls

$L_s = \{(ID_j, \alpha_j, PID_j)\}$.

HEX

4.3. Drone registration phase

In this phase, Drone submits its identity to control server CS and get its secret key. The detailed steps are as shown in Fig. 4.

1. V_j randomly selects its identity ID_j and send it with registration request to CS.
2. CS computes $PID_j = h(ID_j \| k)$, $\alpha_j = h(ID_j \| MSK)$ and stores (ID_j, α_j, PID_j) in list L_s securely. Finally, CS sends (α_j, PID_j) to V_j via a secure channel.
3. V_j receives (α_j, PID_j) and stores them securely.

Output:

```
--- Drone Registration Phase ---
IDj = droneID44233
PIDj and alpha_j from CS (HEX)
PIDj = 32422a157daf648d008a3c9f64458b45a77d7bce4cf86c792f8daaf5acc3263a
alpha_j = bcfe90f36f93afb2e056b5b513b04548c58e0a37a43cdc1bb73197900b79719f
```

```java
//User Registration Phase
    System.out.println("\n--- User Registration Phase ---");

    //User sets its identity IDi and Password PWi
    BigInteger usid = new BigInteger(16,rnd);
    IDi = "userID"+usid.toString();
    System.out.println("IDi = "+IDi);

    BigInteger pwi = new BigInteger(40,rnd);
    PWi = "user"+pwi.toString(16);
    System.out.println("PWi = "+PWi);

    //CS computes PIDi and alpha_i for User
    PIDi = Hash(IDi+k.toString());
    alphai = Hash(IDi+MSK.toString());
    System.out.println("\nPIDi and alpha_i from CS (HEX)");
    System.out.println("PIDi = "+PIDi.toString(16));
    System.out.println("alpha_i = "+alphai.toString(16));

    //CS stores (IDi, alpha_i, PIDi) in Ls
    Ls[1][0] = IDi;
    Ls[1][1] = alphai.toString();
    Ls[1][2] = PIDi.toString();

    //User alpha^m_i and PID^m_i
    BigInteger tmp = Hash(IDi+PWi);
    alphaim = tmp.xor(alphai);
    PIDim = tmp.xor(PIDi);
    System.out.println("\nUser PID^m_i and alpha_m_i (HEX)");
    System.out.println("PID^m_i = "+PIDim.toString(16));
    System.out.println("alpha^m_i = "+alphaim.toString(16));
```

①

α^m_i

PID^m_i

4.2. User registration phase

In this phase, user U_i joins the IoD environment, registers on control server CS and gets his/her secret key via a secure channel. The computation steps are as shown in Fig. 3.

1. U_i first randomly selects his/her identity ID_i and password PW_i, then sends ID_i with registration request to CS.
2. Upon receiving the message from U_i, CS computes $PID_i = h(ID_i \parallel k)$, $\alpha_i = h(ID_i \parallel MSK)$ and stores (ID_i, α_i, PID_i) in list L_s securely. Then, CS sends (α_i, PID_i, PID_j) to U_i via a secure channel.
3. U_i receives (α_i, PID_i, PID_j) and computes $\alpha^m_i = h(ID_i \parallel PW_i) \oplus \alpha_i$, $PID^m_i = h(ID_i \parallel PW_i) \oplus PID_i$. Finally, U_i stores $(\alpha^m_i, PID^m_i, PID_j)$ securely.

Output :

```
--- User Registration Phase ---
IDi = userID59676
PWi = user55a34d5331

PIDi and alpha_i from CS (HEX)
PIDi = f9bcea38b0a277fb6ff25db7809575e8e2005f50cb7afd5678dd34762363988e
alpha_i = 72faf0fd6d1dc8a6dbbb949f1c667c50acea2fac8769e73400aa49b85be17001

User PID^m_i and alpha_m_i (HEX)
PID^m_i = 11a596226d852d4a611a0152b30ef8175a2d9060ae1240e636ce0d8a3279f0
alpha^m_i = 8b57bf53ffd23a70fe28d329ce40074059b05d6c2cbd08229e41b3c3f2b0917f
```

$L_s = \{(ID_j, \alpha_j, PID_j), (ID_i, \alpha_i, PID_i)\}$.

PID'_i

```java
//---4.4.(1)
    //User Ui computes PIDi and alpha_i
    BigInteger z = Hash(IDi+PWi);
    PIDi = PIDim.xor(z);
    alphai = alphaim.xor(z);

    //User Ui chooses 160 bits r1
    BigInteger r1 = new BigInteger(160,rnd);
    //User Ui sets current timestamp ST1
    LocalTime ST1 = LocalTime.now();
    System.out.println("Current time stamp ST1 = "+ST1);

    //User Ui computes M1,M2,M3,M4
    BigInteger M1,M2,M3,M4;
    M1 = Hash(PIDs.toString()+ST1.toString()).xor(PIDi);
    M2 = Hash(PIDi.toString()+PIDs.toString()+alphai.toString()).xor(r1);
    M3 = Hash(PIDi.toString()+PIDs.toString()+alphai.toString()+r1.toString()).xor(PIDj);
    M4 = Hash(PIDi.toString()+PIDj.toString()+PIDs.toString()+alphai.toString()+r1.toString());
```

$z = h(ID_i \parallel PW_i)$

ST_1

1. U_i first inputs his/her identity ID_i and password PW_i, and the mobile will compute $PID_i = PID^m_i \oplus h(ID_i \parallel PW_i)$, $\alpha_i = \alpha^m_i \oplus h(ID_i \parallel PW_i)$. Then it randomly chooses a 160 bits number $r_1 \in Z^*_n$ and the current timestamp ST_1 to calculate the following. Finally, it sends authentication request message (M_1, M_2, M_3, M_4) to CS through a public channel.

$= z$

$$M_1 = h(PID_s \parallel ST_1) \oplus PID_i$$
$$M_2 = h(PID_i \| PID_s \| \alpha_i) \oplus r_1$$
$$M_3 = h(PID_i \| PID_s \parallel \alpha_i \| r_1) \oplus PID_j$$
$$M_4 = h(PID_i \| PID_j \| PID_s \| \alpha_i \| r_1)$$

$M_4 = h(PID_i \| PID_j \| PID_s \| \alpha_i \| r_1)$

```java
//---4.4.(2)
    BigInteger PIDip, alphaip, r1p, PIDjp, M4p;

    //CS check the validation of time
    long timeThreshold = 3; //maximum time threshold
    System.out.println("CS checks validation time");
    System.out.println("Max time threshold deltaT = "+timeThreshold+" second");
    LocalTime time = LocalTime.now();
    System.out.println("Time Now = "+time);
    Duration dT = Duration.between(ST1, time); // compute time-ST1

    long deltaT = dT.getSeconds();

    //Check if deltaT > timeThreshold
    if (deltaT>timeThreshold) {
        System.out.println("CS rejects the authentication request");
        return;
    } else {
        System.out.println("CS accepts the messages");
    }

    //CS computes PID'i
    PIDip = M1.xor(Hash(PIDs.toString()+ST1.toString()));

    //CS retrieves a'i from PID'i in Ls
    alphaip = getAlpha(PIDip, Ls);
    //If the a'_i = 0 then PID'_i is not valid
    if (alphaip.toString()=="0") {
        System.out.println("The identity PIDi' is not found in Ls");
        return;
    }

    //CS computes r1', PIDj', M4'
    r1p = M2.xor(Hash(PIDip.toString()+PIDs.toString()+alphaip.toString()));
    PIDjp = M3.xor(Hash(PIDi.toString()+PIDs.toString()+alphaip.toString()+r1p.toString()));
    M4p = Hash(PIDi.toString()+PIDjp.toString()+PIDs.toString()+alphaip.toString()+r1p.toString());
```

Compute validation of time.

PID'_i

$\alpha'_i = \alpha_i$ from L_s

search α_i

r'_1

2. After receiving the authentication request message (M_1, M_2, M_3, M_4) from U_i, CS first checks the validation of time by $time - ST_1 \leq \Delta T$, in which ΔT is the maximum time threshold of accepting messages and $time$ is the current time received message. If it is true, CS goes to the next step; Otherwise, CS rejects the authentication request. CS further computes $PID'_i = M_1 \oplus h(PID_s \parallel ST_1)$ and retrieves α'_i in the list L_s. Then CS computes the following.

$$r'_1 = M_2 \oplus h(PID'_i \| PID_s \| \alpha'_i)$$
$$PID'_j = M_3 \oplus h(PID'_i \| PID_s \parallel \alpha'_i \| r'_1)$$
$$M'_4 = h(PID'_i \| PID'_j \| PID_s \| \alpha'_i \| r'_1)$$

Output :

```
--- Authentication Phase ---
Current time stamp ST1 = 00:08:12.419601400
CS checks validation time
Max time threshold deltaT = 3 second
Time Now = 00:08:12.519264400
CS accepts the messages
```

Find α from L_s with PID

α_i from L_s.

```java
private static BigInteger getAlpha(BigInteger PID, String[][] LS) {
    //This method to get alpha from its PID in Ls
    String alpha="0";
    for (int i=0;i<LS.length;i++) {
        if (PID.toString().equals(LS[i][2])) {
            alpha=LS[i][1];
        }
    }
    return new BigInteger(alpha);
```

```java
//---4.4.(3)
    //CS checks M4 = M4'
    System.out.println("\nCS checks for M4");
    if (M4.equals(M4p)) {
        System.out.println("M4 = "+M4.toString(16));
        System.out.println("M4' = "+M4p.toString(16));
        System.out.println("Verification status : "+"M4 = M4'");
    } else {
        System.out.println("M4 = "+M4.toString(16));
        System.out.println("M4' = "+M4p.toString(16));
        System.out.println("Verification status: "+"M4 != M4'");
        return;
    }

    //CS retrieves aj' from PIDj' in Ls
    BigInteger alphajp = getAlpha(PIDjp, Ls);
    //If the a'j = 0 then PID'j is not valid
    if (alphajp.toString()=="0") {
        System.out.println("The identity PID'j is not found in Ls");
        return;
    }

    //CS computes M5,M6,M7
    BigInteger M5,M6,M7;
    M5 = Hash(PIDjp.toString()+alphajp.toString()).xor(r1p);
    M6 = Hash(PIDjp.toString()+PIDs.toString()+alphajp.toString()+r1p.toString()).xor(PIDip);
    M7 = Hash(PIDip.toString()+PIDjp.toString()+PIDs.toString()+alphajp.toString()+r1p.toString());
```

$M_4 = M_4'$?

STOP

Find α_j' in L_s.

STOP

3. CS checks the validation of $M_4' = M_4$. If they are equal, CS can authenticate U_i and retrieves α_j' in the list L_s through PID_j', then continue to do the following steps. Otherwise, CS rejects the authentication request. Finally, CS sends message (M_5, M_6, M_7) to V_j through a public channel.

$$M_5 = h(PID_j' \parallel \alpha_j') \oplus r_1'$$
$$M_6 = h(PID_j' \| PID_s \parallel \alpha_j' \| r_1') \oplus PID_i'$$
$$M_7 = h(PID_i' \| PID_j' \| PID_s \| \alpha_j' \| r_1')$$

Output:

```
CS checks for M4
M4 = faa05b5fa5c6841ab75995241c72bad690422543bfdf3e6735f94bd9e5d8de05
M4' = faa05b5fa5c6841ab75995241c72bad690422543bfdf3e6735f94bd9e5d8de05
Verification status : M4 = M4'
```

```java
//---4.4.(4)
    //Drone Vj computes r1'', PIDi'', and M7'
    BigInteger r1pp, PIDipp, M7p;
    r1pp = M5.xor(Hash(PIDj.toString()+alphaj.toString()));
    PIDipp = M6.xor(Hash(PIDj.toString()+PIDs.toString()+alphaj.toString()+r1pp.toString()));
    M7p = Hash(PIDipp.toString()+PIDj.toString()+PIDs.toString()+alphaj.toString()+r1pp.toString());
```

r_1''

PID_i''

M_7'

4. After receiving message (M_5, M_6, M_7) from CS, V_j first computes the following:

$$r_1'' = M_5 \oplus h(PID_j \parallel \alpha_j)$$
$$PID_i'' = M_6 \oplus h(PID_j \| PID_s \parallel \alpha_j \| r_1'')$$
$$M_7' = h(PID_i'' \| PID_j \| PID_s \| \alpha_j \| r_1'')$$

Drone V_j

```java
//---4.4.(5)
    //Drone Vj check M7'=M7
    System.out.println("\nDrone Vj checks for M7");
    if (M7.equals(M7p)) {
        System.out.println("M7 = "+M7.toString(16));
        System.out.println("M7' = "+M7p.toString(16));
        System.out.println("Verification status : "+"M7 = M7'");
    } else {
        System.out.println("M7 = "+M7.toString(16));
        System.out.println("M7' = "+M7p.toString(16));
        System.out.println("Verification status: "+"M7 != M7'");
        return;
    }

    //Drone Vj chooses 160 bits r2
    BigInteger r2 = new BigInteger(160, rnd);

    //Drone Vj computes M8, M9, M10, SKji
    BigInteger M8,M9,M10,SKji;
    M8 = Hash(PIDj.toString()+PIDipp.toString()+r1pp.toString()).xor(r2);
    M9 = Hash(r1pp.toString()+r2.toString());
    M10 = Hash(PIDipp.toString()+PIDj.toString()+PIDs.toString()+r1pp.toString()+r2.toString()+M9.toString());
    SKji = Hash(PIDipp.toString()+PIDj.toString()+PIDs.toString()+M9.toString());

    System.out.println("Session Key SKji = "+SKji.toString(16));
```

$M_7' = M_7$?

r_2

5. V_j checks the validation of $M_7' = M_7$. If it does not hold, V_j rejects the communication request. Otherwise, V_j can authenticate CS and randomly choose a 160 bits number $r_2 \in Z_n^*$, then continue to do the following steps. Finally, V_j sends message (M_8, M_{10}) to U_i through a public channel.

$$M_8 = h(PID_j \| PID_i'' \| r_1'') \oplus r_2$$
$$M_9 = h(r_1'' \parallel r_2)$$
$$SK_{ji} = h(PID_i'' \| PID_j \parallel PID_s \| M_9)$$
$$M_{10} = h(PID_i'' \| PID_j \| PID_s \parallel r_1'' \| r_2 \| M_9)$$

SK_{ji} is common key for drone V_j.

||

SK_{ij} is common key for user U_i

Output:

```
Drone Vj checks for M7
M7 = 72e45b55724fdd49e4634dea37a598b53227b3ccbb1c14c158ee8ad4d6bcb765
M7' = 72e45b55724fdd49e4634dea37a598b53227b3ccbb1c14c158ee8ad4d6bcb765
Verification status : M7 = M7'
Session Key SKji = 9c4f6e541505098a3b32a296e97bf69f8c8a63c42913b3a13dbc53a7cb54e915
```

User U_i

```java
//---4.4.(6)
    BigInteger r2p, M9p, M10p, SKij;
    //User Ui computes r2', M9',M10'
    r2p = M8.xor(Hash(PIDj.toString()+PIDi.toString()+r1.toString()));
    M9p = Hash(r1.toString()+r2p.toString());
    M10p = Hash(PIDi.toString()+PIDj.toString()+PIDs.toString()+r1.toString()+r2p.toString()+M9p.toString());

    //User Ui checks M10' = M10
    System.out.println("\nUser Ui checks for M10");
    if (M10.equals(M10p)) {
        System.out.println("M10 = "+M10.toString(16));
        System.out.println("M10' = "+M10p.toString(16));
        System.out.println("Verification status : "+"M10 = M10'");
    } else {
        System.out.println("Verification status: "+"M10 != M10'");
        return;
    }

    //User Ui calculates the common session key SKij
    SKij = Hash(PIDi.toString()+PIDj.toString()+PIDs.toString()+M9p.toString());
    System.out.println("Session Key SKij = "+SKij.toString(16));
```

M_9'

check $M_{10}' = M_{10}$

STOP

6. When U_i receives message (M_8, M_{10}) from V_j, he/she first computes as the follows. U_i checks the validation of $M_{10}' = M_{10}$. If they are equal, U_i can authenticate V_j and calculate the common session key $SK_{ij} = h(PID_i \| PID_j \parallel PID_s \| M_9') = SK_{ji}$. Otherwise, U_i rejects the communication request.

$$r_2' = M_8 \oplus h(PID_j \| PID_i \| r_1)$$
$$M_9' = h(r_1 \parallel r_2')$$
$$\cancel{M_{10}' = h(PID_i \| PID_j \| PID_s \| r_1 \| r_2')} \times$$
$$SK_{ij} = h(PID_i \| PID_j \parallel PID_s \| M_9')$$

$M_{10}' = h(PID_i \| PID_j \| PID_s \| r_1 \| r_2' \| M_9')$

Drone V_j : SK_{ji}

User U_i : SK_{ij}

```
User Ui checks for M10
M10 = ad3f84765d47b7d1fdab449b00bc1895f3f2239b70d040dc5ae7c7f6a4a28804
M10' = ad3f84765d47b7d1fdab449b00bc1895f3f2239b70d040dc5ae7c7f6a4a28804
Verification status : M10 = M10'
Session Key SKij = 9c4f6e541505098a3b32a296e97bf69f8c8a63c42913b3a13dbc53a7cb54e915
```

check $SK_{ij} = SK_{ji}$?

```
---- Conclusion ----
User Ui and Drone Vj using same session key (SKij = SKji)
9c4f6e541505098a3b32a296e97bf69f8c8a63c42913b3a13dbc53a7cb54e915
```

```java
    System.out.println("\n\n---- Conclusion ----");
    if (SKij.equals(SKji)) {
        System.out.println("User Ui and Drone Vj using same session key (SKij = SKji)");
        System.out.println(SKij.toString(16));
    } else {
        System.out.println("User and Drone have different session key (SKij != SKji)");
    }
```