**UNIVERSITY OF INFORMATION TECHNOLOGY**

**FALCUTY OF COMPUTER NETWORKS AND COMMUNICATIONS**

----------ꕥ----------

**FINAL REPORT**

# "ECC-CoAP: Elliptic Curve Cryptography Based Constraint Application Protocol for Internet of Things"

Instructor: **Ph.D. Nguyễn Ngọc Tự**

Group members:

**Lã Ngọc Ánh – 20521065**

**Trần Thúy Anh – 20521085**

**Trương Đình Trọng Thanh – 20520766**

Course: **NT219.M21.ATCL**

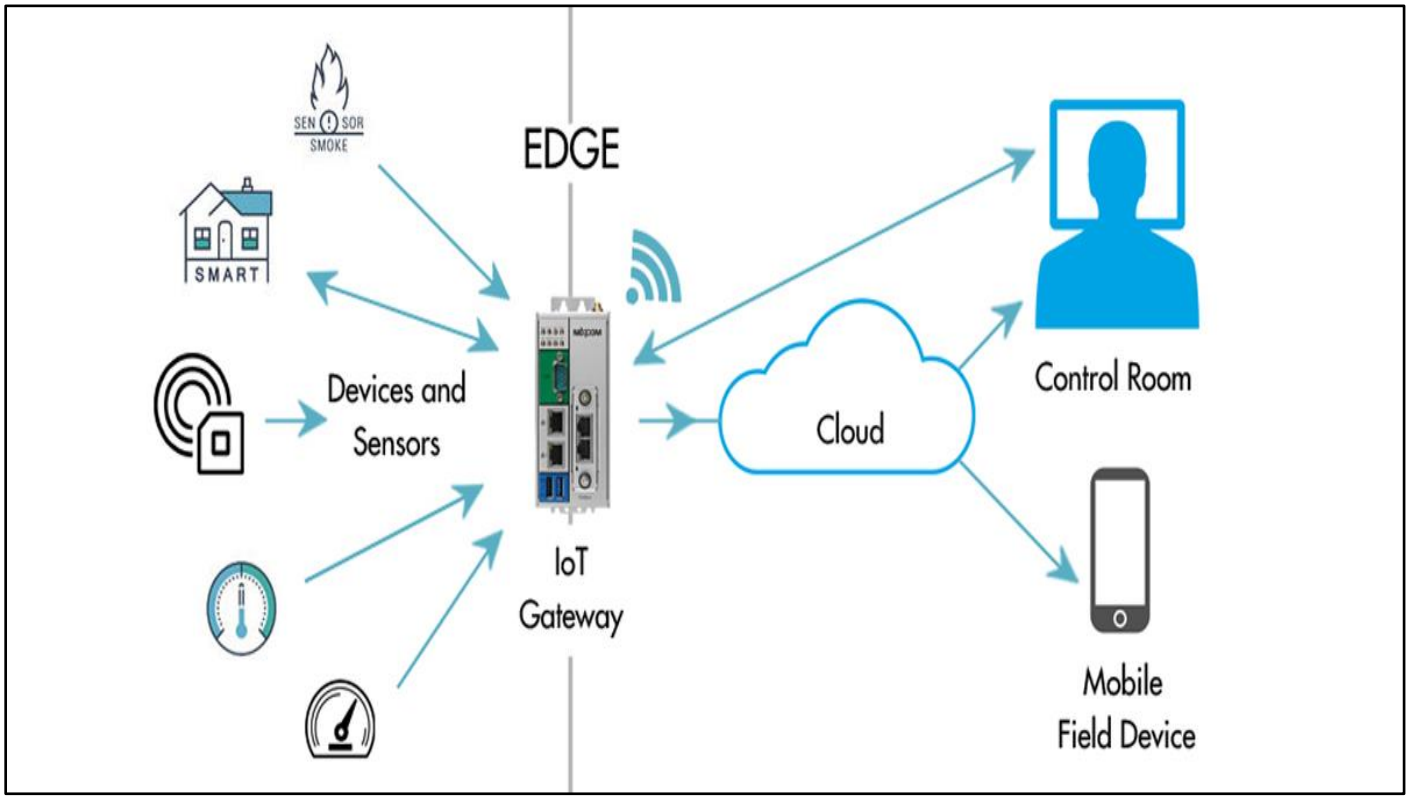
Hồ Chí Minh, 16/06/2022

## Table of contents:

## Scenario:

In application layer, the CoAP is mainly used for secure communication between the constraint smart IoT devices and server.

The CoAP protocol is generally associated with connectionless User Datagram Protocol (UDP) and works based on Representational State Transfer architecture. However, several limitations regarding the key management, session establishment and multi-cast message communication within the DTLS layer are present in CoAP. Hence, development of an efficient protocol for secure session establishment of CoAP is required for IoT communication. Thus, to overcome the existing limitations related to key management and multicast security in CoAP, we have proposed an efficient and secure communication scheme to establish secure session key between IoT devices and remote server using lightweight elliptic curve cryptography (ECC). The proposed ECC-based CoAP is referred to as ECC-CoAP that provides a CoAP implementation for authentication in IoT network. A number of well-known crypto-graphic attacks are analyzed for validating the security strength of the ECC-CoAP and found that all these attacks are well defended. The performance analysis of the ECC-CoAP shows that our scheme is lightweight and secure.




## Research motivations:

- CoAP uses UDP, which isn't a trusted protocol, creating unordered message through transport

- UDP's lack of a verification mechanism and end-to-end connections

- It acknowledges each message receipt and thus increases processing time. Furthermore, it does not verify whether the received message was properly decrypted;

- Having several limitations regarding the key management, session establishment an multi-cast message communication within the DTLS layer are present in CoAP.

=> We have proposed an efficient and secure communication scheme to establish secure session key between IoT devices and remote server using lightweight elliptic curve cryptography (ECC) combine with LESS protocol called "CoAP-ECC"

## Proposed scheme:

### Pre-requisite of ECC-CoAP

Initially, the server selects an elliptic curve $Ep(a,b)$ over a prime finite field $Fp$, where $P$ is the generator of order $n$. Next, the private key as $q_S \in \mathbb{Z}* p$ selected by the server and calculates its public key as $Q_S=q_S.P$ using ECC based scalar point multiplication (ECPM). Similarly, user/IoT device randomly selects a large random number $q_U \in \mathbb{Z}* P$ such as $0< q_U< n$ as a private key of the user/IoT device and generates the public key $Q_U$ as $Q_U=q_U.P$. The user/IoT device then gets the ECC based public key certificate $CA_U$, combining its identity $ID_U$ and public key $Q_U$ from the certificate authority $CA$.



**Table 2** Notations and respective descriptions

| Notation | Description |
|---|---|
| $U$ | User/IoT device |
| $S$ | Server |
| $F_p$ | A large prime finite field over $p$ |
| $E_p(a,b)$ | An elliptic curve is defined on $F_p$ |
| $P$ | A generator point on $E/F_p(a,b)$ with order $n$ |
| $ID_U$ | User's identity |
| $PW_U$ | Random password selected by the user |
| $DID_U$ | Dynamic login identity of the user/device generated by server |
| $K$ | Common key used for Encryption/Decryption for both user/ IoT device and server end |
| $S_K$ | Dynamic session identity generated from server end |
| $r_U$ | Random number selected by the user |
| $r_S$ | Random number selected by the server |
| $R_U$ | Random value produced by the user where $R_U=r_U.Q_U$ |
| $R_S$ | Random value produced by the server where $R_S=r_S.Q_S$ |
| $h(.)$ | One way secure hash function such as $SHA1$ |
| $E/D$ | Symmetric encryption/decryption algorithm |
| $(q_S, Q_S)$ | Private–public key pair of server where $Q_S=q_S.P$ |
| $(q_U, Q_U)$ | Private–public key pair of user/IoT device where $Q_U=q_U.P$ |
| $\|$ | Concatenation |

### Working Procedures of ECC-CoAP

The detail step-wise working procedures of ECC-CoAP for communication between the user/IoT device and server is shown in Fig. 1 and illustrated below where $X \rightarrow Y: M$ denotes that sender $X$ sends a message $M$ to receiver $Y$.

***Step 0*: U → S:$ID_U$, $CA_U$, $EK_X(H_U)$, $T_1$**

Initially, user/IoT device generates a random high entropy password $PW_U$. Then user/ IoT device computes (i) the symmetric shared key $K$ between user/IoT device and server as $K = q_U.QS = q_U.q_S.P = (K_X,K_Y)$ where $q_U$ and $Q_S$ are the private key of user/IoT device and public key of server respectively, and (ii) $H_U = h(ID_U||PW_U||q_U)$where $h$ is a one way irreversible cryptographic hash function and encrypts $H_U$ using $K_X$. Finally, it sends a *session initiation request* containing $ID_U$, $CA_U$, encrypted $H_U$ and $T_1$ to server.

***Step 1*: S → U:$ID_S$,$EK_X(DID_U||R_S)$,$T_2$**

After receiving the session initiation request from user/IoT device in time $T_2$, server checks $|T_2 - T_1| \leq \Delta T$? If yes the server retrieves the user's identity $ID_U$ and public key $Q_U$ from $CA_U$ and checks retrieved $ID_U$=received $IDU$? If fails the communication is terminated; Otherwise, the server (i) calculates the symmetric shared key $K=qS.QU=qS.qU.P=(K_X,K_Y)$,(ii) decrypts the encrypted message using $K_X$ and gets $H_U$, (iii) generates a dynamic identity of the user/IoT Device $DID_U = h(ID_U||K||H_U)$,(iv) selects a random number $r_S \in \mathbb{Z}* p$ to calculate the server's random point $R_S = r_S.Q_S = r_S.q_S.P$ using ECPM,(v) stores $H_U$ and $DID_U$ at the server's database for future reference,(vi) concatenates $DID_U$ and $R_S$, then the concatenated message is encrypted using symmetric key $K_X$ and finally (vii) sends the $ID_S$, encrypted message and $T_2$ to IoT device as *server challenge.*

***Step 2*: U → S:$R_U$,$ESK_X(M_U)$, $T_3$**

The IoT device receives the server's challenge in time $T_3$ and verifies the legitimacy of the server's challenge i.e. checks $|T_3 - T_2| \leq \Delta T$? If yes, the IoT device decrypts the encrypted server challenge using $K_X$ and gets $DID_U$ and $R_S$. Now, it(i) calculates dynamic identity $DID_U = h(ID_U||K||H=)$and (ii) compares the calculated $DID_U$ with received $DID_U$. If the comparison is unsuccessful the communication is terminated; otherwise, the IoT device selects a random number $r_U \in \mathbb{Z}* p$ and calculates a random point $R_U = r_U.Q_U = r_U.q_U.P$. It then calculates the session key as $SK = q_U.r_U.R_S = qU.rU.rS.qS..P = (SK_X,SK_Y)$ and $M_U = h(own\ R_U||own\ H_U||DID_U||T_3)$. Now it encrypts $M_U$ using the recently calculated session key $SK_X$ and sends the encrypted message with $R_U$ and $T_3$ as a response to server's challenge. A variable *count* is initialized with *0* and incremented with *1* after each unsuccessful response message transmission. Each IoT device is allowed to get *3* attempts to authenticate to server otherwise the device will be blocked for a specific period of time. This method is implemented to stop cryptographic attacks like brute-force attack.

***Step 3*: S → U: $ESK_X(M_S)$, $T_4$**

$SK = q_S.r_S\ .R_U = q_S.r_S.r_U.q_U.P = q_U.r_U.r_S.q_S.P = (SK_X,SK_Y)$, (ii) decrypts the encrypted client challenge and gets $M_U$ as $DSK_X(ESK_X(M_U)) = M_U$ (iii) calculates $M_U$ /= $h(received\ R_U||Stored\ H_U||Stored\ DID_U||T_3)$and (iv) checks $MU$ /= $MU$? If both are equal, the IoT device is authenticated to server. Now the server calculates $MS = h(RS||MU)$, encrypts $MS$ using session key $SKX$ and finally sends the encrypted $MS$ and the current timestamp $T4$ as a server's response to IoT device.

***Step 4:* U → S : *Message communication M is done in EXI format***

The IoT device receives the server response in time $T_5$ and checks $|T_5 - T_4| \leq \Delta T$? If yes, IoT device decrypts the encrypted server's response using session key $SK_X$ and gets $M_S$ as $DSK_X(ESK_X(M_S)) = M_S$. Now it calculates $M_S$ /=$h(received\ R_S||sent\ M_U)$ and checks $M_S$ /= $M_S$? If both are equal, the server is authenticated to IoT device; otherwise the communication is terminated. All the further message communication $M$ is done in $EXI$ format using $SK_X$ between the server and IoT device.

## Security Analysis

### 1. Informal Security Analysis

This section illustrates informal security analysis of ECC-CoAP protocol using mathematical procedures. Some practical assumptions are taken into account for proving the security strength of the protocol as given in the literature.

### 1.1 Man-in-the-Middle Attack

Let an adversary $\tilde{A}$, present between user/IoT device and server, intercepts the session initiation message containing $ID_U$,$CA_U$,$EK_X(H_U)$, $T1$ and intends to modify it in such a way that it seems to be coming from a legitimate user containing valid identity $ID_U$ of the legitimate user but with the replaced value of $CA_U$ and $H_U$ of the adversary. However, after receiving the message, server retrieves $ID_U$ from $CA_U$ and checks retrieved $ID_U$=received $ID_U$?. It results failed verification and communication will be terminated. Moreover, if the adversary $\tilde{A}$ only tries to modify the parameter HU it will not be possible as it is communicated by encrypting using ECDH based contributory symmetric key which is hard to forge in polynomial time. Hence, the ECC-CoAP scheme is robust against Man-in-the-Middle Attack.

### 1.2 Denial-of-Service (DoS) Attack

In the client response and challenge phase of ECC-CoAP scheme, if the IoT device fails to be authenticated by server within three attempts then the IoT device will be blocked for a specific period of time. A variable *count* is initialized with 0 and incremented by 1 after each of the unsuccessful response message transmission by the server. Every IoT device gets at most 3 attempts to be authenticated. Hence, an adversary $\tilde{A}$ will not be able to send multiple fuzzy requests (more than three) to make the system resource overloaded to make the services unavailable to the legitimate user, thus ECC-CoAP restricts the DoS attack.

### 1.3 Replay Attack

In the client response and challenge phase of the proposed ECC-CoAP scheme, if an adversary $\tilde{A}$ acquires the authentication message of the user $\{R_U, ESK_X(M_U), T_3\}$ where $M_U=h(R_U||H_U||DID_U||T_3)$ and tries to replay it in later session just changing the current recorded time from $T_3$ to $T_3'$ $\{RU, ESKX(MU), T3\}$. After receiving this authentication request by the server, it will check $||T_4 - T_3'|\leq \Delta T$, which would be successful. However, after checking the timestamp it will calculate $M_U' = h(R_U||H_U||DID_U||T_3')$ which will not be same as the received $M_U$. Hence, the session will be terminated. As in the proposed scheme, current timestamp is not only sent as a parameter of the message it also included as a parameter of $M_U$ it is resilient to reply attack.

### 1.4 Insider Attack

Users provide their valid credentials to be authenticated to the remote server by assuming the remote server is trusted. However, sometimes it is noted that any insider of the remote server acts as an adversary $\tilde{A}$ after getting some crucial credentials of the user stored into the remote server. In proposed ECC-CoAP, the server stores $H_U$ and $DID_U$ as the crucial credentials for further authentication of IoT device. In this scenario, if $H_U$ and $DID_U$ are acquired by the insider, still it cannot be authenticated as a legitimate user. For generating a valid authentication request, it is required to generate a random nonce say $R_U'=r_U'.Q_U$ and $M_U'=h(R_U'||H_U||DID_U||T_3)$.Then $M_U'$ is encrypted using $SK$ where $SK$ is ECDH based session key calculated as $SK= q_U.r_U.R_S$ where $q_U$ is the private key of the valid user. So, it is impossible for the insider to somehow calculate the session key $SK$ due to hardness of ECC as well as it includes private key of the valid user. Hence ECC-CoAP is safe against insider attack.

### 1.5 User Impersonation Attack

If an adversary $\tilde{A}$ pretends to be an authorized user of the system. The adversary $\tilde{A}$ impersonates the transmitted message and re-transmits it pretending as a valid user. User impersonation attack cannot be possible in client side due to the following reasons:(i) At the time of session initiation, user/IoT device sends the session initiation message$\{ID_U, CA_U,EK_X(H_U), T_1\}$ to server. If the identity of the IoT device is modified then the server can easily track it from the ECC based public key certificate $CA_U$ (containing identity $ID_U$ and public key $Q_U$) as it is certified from the certificate authority and cannot be forged. Moreover, hash digest of the identity of the user $H_U$ (containing identity $ID_U$, password $PW_U$ and private key $q_U$) cannot be replaced by the adversary $\tilde{A}$ as it is transmitted in

encrypted form by using the symmetric key $K_X$. However, $K_X$ cannot be calculated due to hardness of ECDLP. So, $H_U$ cannot be decrypted. (ii) In client's response and challenge phase of ECC-CoAP, user/IoT device sends authentication request message containing $\{R_U, ESK_X(M_U), T_3\}$.If the adversary $\tilde{A}$ intends to generate the masked identity of the user $M_U$ it will not be able to compute it as it is encrypted using $SK$ which is ECDH based session key where $SK=q_U r_U. r_S.q_S.P$ composed of private of the user $q_U$. Hence, proposed ECC-CoAP is CoAP scheme is robust against user impersonation attack.

### 1.6 Server Impersonation Attack

In this type of attack, an adversary $\tilde{A}$ acts as a server by knowing some secret credentials of the server and further communicates with the user to exchange the messages. At first, the server sends a challenge message $ID_S, EK_X(DID_U || R_S)$, $T_2$ as a response of the session initiation request $\{ID_U, CA_U, EK_X(H_U), T_1\}$ of the user/IoT device. However, to forge the server challenge to user the adversary $\tilde{A}$ needs to decrypt the value of $HU$ to compute valid $DID_U=h(ID_U||K||H_U)$using the symmetric key $K_X$. However, $K$ is tough to compromise due to the hardness of ECDLP. So, the adversary $\tilde{A}$ cannot be able to determine the dynamic identity of IoT device valid $DID_U$. So, ECC-CoAP is safe against server impersonation attack.

### 1.6 Offline Password Guessing Attack

This is one of the most popular attacks that mainly occur at the password based authentication schemes due low entropy passwords chosen by the user. So, a strong password based scheme can restrict this type of attack. In ECC-CoAP, password $PW_U$ is only used to calculate $H_U$ where $H_U=h\ (ID_U||PW_U||q_U)$ which stored for further communication. Hence, the adversary $\tilde{A}$ cannot be able to generate $HU$ only by randomly guessing the password the user/IoT device as $H_U$ requires $q_U$, the private key of the user. Thus, ECC-CoAP is protected against offline password guessing attack.

### 1.7 Known Session Specific Temporary Attack

To avoid the occurrence of known session-specific temporary information attack, session key in ECC-CoAP is calculated in IoT device end as $SK= q_U.r_U.R_S = q_U.r_U.r_S.q_S.P$ and from server end as $SK=q_S.r_S.R_U= q_S.r_S.r_U.q_U.P=q_U.r_U.r_S.q_S.P$ which contains the private keys of each end. Although any one of the secret random values like $r_s$ or $r_U$ of the server and user respectively are accidentally exposed to adversary $\tilde{A}$, still the session key cannot be generated due to the unavailability of the private keys. So, ECC-CoAP is free from known session specific temporary attack.

### 1.8 Session Key Computation Attack

ECC-CoAP is designed to agree upon a common secret session key $SK=q_U.r_U.R_S =q_S.r_S.R_U= q_U.r_U.r_S.q_S.P$ to carry out further data exchange securely between the user/IoT device and server. The proposed scheme provides ECDLP based secure session key which is hard to compromise due to hardness of ECDLP. Further, the session key cannot be computed it is generated based on two private keys and two random numbers both from user/IoT device and server end. If any of the secret parameters are somehow guessed or acquired in polynomial time, the other parameters are not available to the adversary $\tilde{A}$ for session key computation. Hence, ECC-CoAP is resilient to session key computation attack.

### 1.9 Efficient Mutual Authentication

ECC-CoAP provides a mutual authentication between the user/IoT devices and server based on two secret credentials $M_U$ and $M_S$ which are calculated based on secret values, mutually shared between them. During client authentication the server receives $M_U$ encrypted using negotiated session key $SK_X$. $M_U$ /is then calculated by the server using stored parameters $DID_U$ and $H_U$ as $M_U'=h(received\ R_U\ ||Stored\ H_U||Stored\ DID_U||T_3)$. If $M_U$ /and $M_U$ are equivalent then only the user/IoT device is authenticated. Similarly, during server authentication $M/S$ is

calculated by the user/IoT device $M'_S = M'_S = h(received\ R_S||sent\ M_U)$. If $M'S$ and $M_S$ are equivalent then only the server is authenticated. From the above discussion it is clear both server and client validate each other with the prior knowledge as well as received values. So, ECC-CoAP comprises of efficient mutual authentication.

### 1.10 Non-repudiation

Non-repudiation is a property which prevents a sender or entity from denying sending a message to the receiver. Use/IoT device sends the session initiation message containing $\{ID_U, CA_U, EK_X(H_U), T_1\}$ to server. As the message contains the public key certificate of the message includes public key certificate containing valid identity of the user/IoT device it cannot deny about the sending of the message. On the other hand, in server challenge phase, server sends the reply composed of $\{ID_S, EK_X(DID_U|| R_S), T_2\}$ to user/IoT devices with server identity $ID_S$. So, in case also the server cannot deny the sending of message. So, ECC-CoAP comprises of non-repudiation.

### 1.11 Perfect Forward Secrecy

In the proposed ECC-CoAP the symmetric contributory key $K$ is compromised the adversary $\tilde{A}$ cannot calculate the session key SK where $SK = q_U.r_U.R_S = q_S.r_S.R_U = q_U.r_U.r_S..q_S.P$ because with the knowledge of symmetric key $K$ the adversary $\tilde{A}$ does not know the secret private key $(q_U, q_S)$ or the random number of the particular session $(r_U., r_S)$. Even if the adversary $\tilde{A}$can decrypt the message using the compromised symmetric key $K$ to obtain random nonce $R_U$ and $R_S$ where $R_U = r_U.Q_U$ and $R_S = r_S.Q_S$, it cannot acquire the knowledge of session specific random numbers $(r_U., r_S)$ due to the hardness of ECDLP. So, ECC-CoAP achieves the property of perfect forward secrecy.

## 2. Formal Security Analysis

In formal security analysis we have analyzed the security of ECC-CoAP protocol by using through Burrows–Abadi–Needham (BAN) logic For analyzing security related to key agreement and authentication protocol, BAN logic is most widely used mathematical model.

### BAN Logic Based Authentication Proof

**Table 4** Notations for BAN logic

| Notations | Meanings |
|---|---|
| $X \equiv Y$ | The statement $Y$ is believed by $X$ |
| $X \triangleright Y$ | $X$ sees the statement $Y$ |
| $X \sim Y$ | $X$ once said the statement $Y$, sometimes ago |
| $X \Rightarrow Y$ | $X$ has got jurisdiction over $Y$ |
| $\#Y$ | The message $Y$ is taken to be fresh |
| $\langle U \rangle V$ | The formulae $U$ is used in combination of formulae $V$ |
| $(U,V)$ | $U$ or $V$ being part of message $(U,V)$ |
| $\{U,V\}K$ | $U$ or $V$ is encrypted with symmetric key $K$ |
| $\langle U,V \rangle_{k \mapsto X}$ | $U$ or $V$ is encrypted with public key $K$ of $X$ |
| $(U,V)_K$ | $U$ or $V$ is being hashed using key $K$ |
| $X \overset{K}{\rightarrow} Z$ | $X$ and $Z$ can securely contact using shared key $K$ |

**Table 5** Primitive formulae used in BAN logic

| Rules | Definitions |
|---|---|
| Message meaning rule | $\frac{X \equiv X \overset{K}{\longleftrightarrow} Z, X \lhd \langle Y \rangle_Y}{X \equiv Z \mid\sim Y}$ |
| Nonce verification rule | $\frac{B \equiv \#(V), B \equiv D \mid\sim V}{B \equiv D \mid\equiv V}$ |
| Jurisdiction rule | $\frac{B \equiv D \Rightarrow V, B \equiv D \mid\equiv V}{B \mid\equiv V}$ |
| Freshness conjuncatenation rule | $\frac{B \mid\equiv \#(V)}{B \mid\equiv \#(V,T)}$ |
| Belief rule | $\frac{B \mid\equiv (V), B \mid\equiv (T)}{B \mid\equiv (V,T)}$ |
| Session key rule | $\frac{B \mid\equiv \#(V), B \mid\equiv D \mid\equiv V}{B \mid\equiv B \overset{K}{\longleftrightarrow} D}$ |

The concerned following rules and notations of BAN logic are described in, where $X$ and $Z$ are the general instances that participate in a protocol.

Following goals are required to be satisfied by aforesaid rules in order to prove the robustness of the ECC-CoAP under BAN logic.

**Goals**

Goal 1: $S| \equiv S \overset{SK}{\longleftrightarrow} C_{SK}$
Goal 2: $S| \equiv C| \equiv_{SK} S \longleftrightarrow C$
Goal 3: $C| \equiv C \longleftrightarrow S_{SK}$
Goal 4: $C| \equiv S| \equiv C \longleftrightarrow S$

Idealized form of communicated messages

| | |
|---|---|
| Message 1 | $C \rightarrow S : ID_U, H_U, T_1 : \left\{ \langle H_U \rangle_{(ID_U, PW_U, q_U)} \right\}_K$ |
| Message 2 | $S \rightarrow C : ID_S, DID_U, R_S, T_2 : \left\{ \langle DID_U \rangle_{(ID_U, K, H_U)} \right\}_K$ |
| Message 3 | $C \rightarrow S : M_U, R_U, T_3 : \left\{ \langle M_U \rangle_{(R_U, H_U)}, \langle R_U \rangle_{(r_U, q_U, q_S, P)} \right\}_K$ |
| Message 4 | $S \rightarrow C : M_S, T_4 : \left\{ \langle M_S \rangle_{(R_S, M_U)} \right\}_{SK}$ |

Following assumptions are required to authenticate the BAN logic for ECC-CoAP.

$A1: S| \equiv \# T_2, T_4$
$A2: C| \equiv \# T_1, T_3$
$A3: S| \equiv \# q_S$
$A4: S| \equiv \# r_S$
$A5: C| \equiv \# q_U$
$A6: C| \equiv \# r_U$
$A7: S| \equiv C \# Q_U, R_U$
$A8: C| \equiv S \# Q_S, R_S$
$A9: C| \equiv C \overset{K}{\leftrightarrow} S$
$A10: S| \equiv S \overset{K}{\leftrightarrow} C$

**Proof of the Proposed Scheme using BAN Logic**

Message 3

$$C \rightarrow S : M_U, R_U, DID_U, T_3 : \left\{ \langle M_U \rangle_{(R_U, H_U, DID_U)}, \langle R_U \rangle_{(r_U, q_U, P)}, \langle DID_U \rangle_{(ID_U, H_U, K)} \right\}_K$$

By applying seeing rule:

$$S1 : S \triangleleft \left\{ \langle M_U \rangle_{(R_U, H_U, DID_U)}, \langle R_U \rangle_{(r_U, q_U)} \right\}$$

By applying message meaning rule, S1, A10:

$$S2 : S| \equiv C \sim \left\{ \langle M_U \rangle_{(R_U, H_U, DID_U)}, \langle R_U \rangle_{(r_U, q_U, P)} \right\}$$

According to A5, A6, S2 and freshness rule

$$S3 : S| \equiv C| \equiv \#\left\{ \langle M_U \rangle_{(R_U, H_U, DID_U)}, \langle R_U \rangle_{(r_U, q_U, P)} \right\}$$

According to S3, S2 and nonce verification rule

$$S4 : S| \equiv C| \equiv \left\{ \langle M_U \rangle_{(R_U, H_U, DID_U)}, \langle R_U \rangle_{(r_U, q_U, P)} \right\}$$

According to A7, S4 and jurisdiction rule

$$S5 : S| \equiv \left\{ \langle M_U \rangle_{(R_U, H_U, DID_U)}, \langle R_U \rangle_{(r_U, q_U, P)} \right\}$$

As the session key is calculated as

$$SK = q_s.r_S.r_U.q_u.P$$

According to S5, S3 and session key rule

$$S6 : S| \equiv S| \equiv S \overset{K}{\leftrightarrow} C \qquad \text{(Goal 1)}$$

According to S6 and session key rule

$$S7 : S| \equiv C| \equiv S \equiv S \overset{K}{\leftrightarrow} C \qquad \text{(Goal 2)}$$

By applying seeing rule:

$$S8 : C \triangleleft R_S, M_S, T_4 \left\{ \langle R_S \rangle_{(rs, q_U, qs, P)}, \langle M_S \rangle_{(R_S, M_U)} \right\}_K$$

By applying message meaning rule, S8, A10:

$$S9 : C| \equiv S \sim R\left\{ {}_S, M_S, T_4 : \left\{ \langle M_S \rangle_{(R_S, M_U)}, \langle R_S \rangle_{(r_S, r_U, q_U, q_S, P)} \right\}_K \right\}$$

According to A3, A4,S9 and freshness rule

$$S10 : C| \equiv S| \equiv \#\left\{ R_S, M_S, T_4 : \left\{ \langle M_S \rangle_{(R_S, M_U)}, \langle R_S \rangle_{(r_S, r_U, q_U, q_S, P)} \right\}_K \right\}$$

According to S9, S10 and nonce verification rule

$$S11 : C| \equiv S| \equiv \left\{ R_S, M_S, T_4 : \left\{ \langle M_S \rangle_{(R_S, M_U)}, \langle R_S \rangle_{(r_S . r_U . q_U . q_S . P)} \right\}_K \right\}$$

According to A8, S11 and jurisdiction rule

$$S12 : C| \equiv \left\{ R_S, M_S, T_4 : \left\{ \langle M_S \rangle_{(R_S, M_U)}, \langle R_S \rangle_{(r_S . r_U . q_U . q_S . P)} \right\}_K \right\}$$

As the session key is calculated as

$$SK = q_s . r_S . r_U . q_u . P$$

According to S10, S12 and session key rule

$$S13 : C| \equiv C \overset{K}{\leftrightarrow} S \qquad \text{(Goal 3)}$$

According to S13 and session key rule

$$S14 : C| \equiv S| \equiv C \overset{K}{\leftrightarrow} S. \qquad \text{(Goal 4)}$$

**Conclusion**

A flexible ECC based CoAP for communication between the user/IoT device and server for setting up secure session among constraints IoT devices is proposed. The proposed scheme will be used to solve the key management and related security issues of resource constraint IoT devices as well as securely operated in insecure channel. The proposed scheme is mathematically analyzed to show its strong resilience against relevant cryptographic attacks. Moreover, ECC-CoAP is formally verified using well accepted AVISPA simulator and BAN logic and found well secure. Finally, the performance study demonstrates that The scheme is more effective in terms of communication and computation overheads for resource constrained IoT devices. Thus ECC-CoAP becomes cost-effective solution for highly demanded client side IoT based CoAP applications.

**References :**

Majumder, Suman, et al. "ECC-CoAP: Elliptic curve cryptography based constraint application protocol for internet of things." *Wireless Personal Communications* 116.3 (2021): 1867-1896.

**Demo :**

**Outline:**

- Encryption and Decryption is ChaCha 20 (stream cipher)

- For ECDH keys we use ECC (curve 25519)

- Hash algorithm is SHA512

**Demo:** with test case

**Step 0:**

```
Serial.print("Generate random k/f for User/IOT ... ");
Serial.flush();
```

```
//unsigned long start = micros();
Curve25519::dh1(alice_k, alice_f);
```

* User/IOT PublicKey in hex

Qu: 2C 27 45 C2 90 76 F3 D0 A6 D4 74 B7 D6 9B 27 15 47 87 F7 E9 AC 59 5C 65 C8 41 D8 A7 59 B1 CA 4D

* User/IOT PrivateKey in hex

qu: 68 6E E4 39 C3 C0 4D F6 3C AB 4B 70 4A DE AF A5 29 D5 9C D2 96 D7 70 A0 C4 92 EB F0 B2 D1 D7 47

* CAu (ECC Certificate) contain:

+ IDu:ESP8266 (65737038323636 hex)

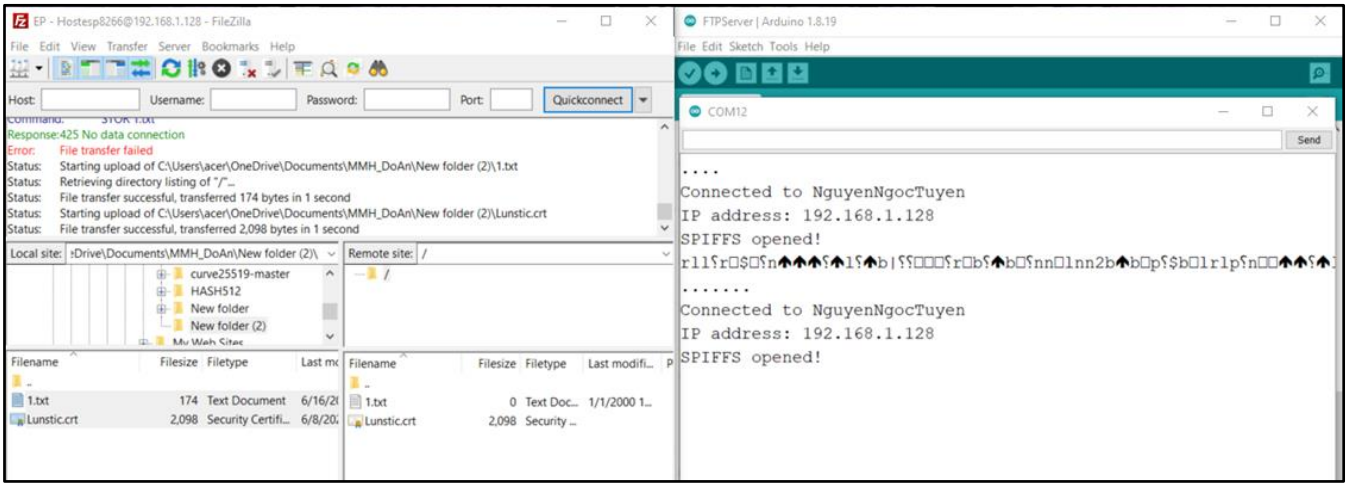
+ Qu: 2C 27 45 C2 90 76 F3 D0 A6 D4 74 B7 D6 9B 27 15 47 87 F7 E9 AC 59 5C 65 C8 41 D8 A7 59 B1 CA 4D

**OPERATION:**

- PWu: 123aA = 3132336141 hex

1) Using FTP connection protocol to transfer file between user and server

User retrieve server public key Qs:

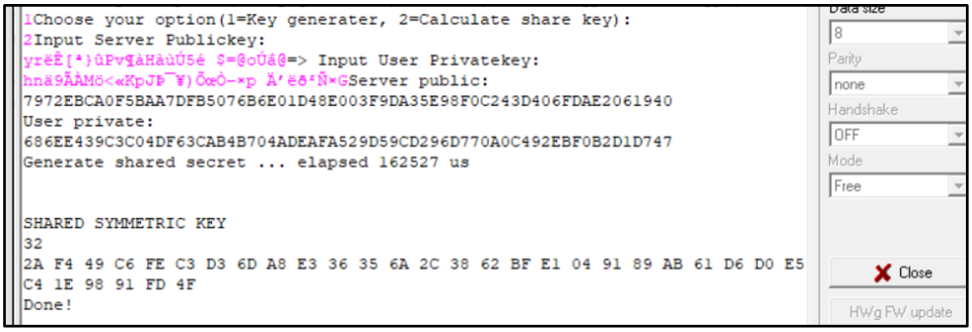



2)

```
Serial.print("Generate shared key ... ");
Serial.flush();
start = micros();
Curve25519::dh2(bob_pub, alice_pri);
```

- Caculate K = qu*Qs = 2A F4 49 C6 FE C3 D3 6D A8 E3 36 35 6A 2C 38 62 BF E1 04 91 89 AB 61 D6 D0 E5 C4 1E 98 91 FD 4F

- Using scalar point multiplication to calculate K




3)

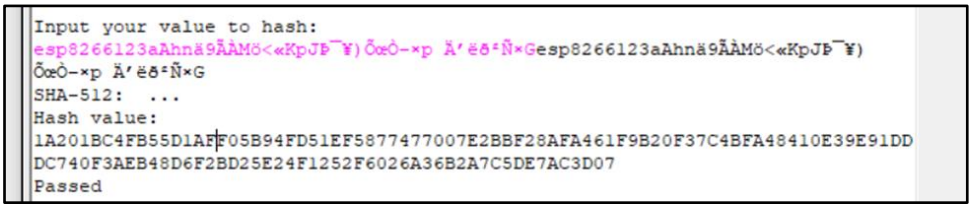
```
for (posn = 0; posn < size; posn += inc)
   {
      len = size - posn;
      if (len > inc)
         len = inc;
      hash->update(test->data + posn, len);
   }
   hash->finalize(value, sizeof(value));
```

Calculate Hu = h(IDu || Pwu || qu) =

Hash value:

1A201BC4FB55D1AFF05B94FD51EF5877477007E2BBF28AFA461F9B20F37C4BFA48410E39E91DDDC740F3AEB48D6F2BD25E24F1252F6026A36B2A7C5DE7AC3D07

(Using SHA512)




4) Encrypt ‘Hu’ using ChaCha20 stream cipher and share key K

```
//====================ENRYPTION============================
   if(i == 1)
   {
   for (posn = 0; posn < test->size; posn += inc) {
      len = test->size - posn;
      if (len > inc)
         len = inc;
```

```
        cipher->encrypt(output + posn, test->plaintext + posn, len);
    }
/====================DECRYPTION=========================
    // phai thay doi output chu y !!
    Serial.println("DECRYPTION: ");
    Serial.println("CIPHER TEXT: ");
    for(int i = 0; i <sizeof(test->ciphertext);++i)
    {
      Serial.printf("%02X", test->ciphertext[i]);
    };
    Serial.println();


    for (posn = 0; posn < test->size; posn += inc) {
        len = test->size - posn;
        if (len > inc)
            len = inc;
        cipher->decrypt(output + posn, test->ciphertext + posn, len);
    }
```

$E_{kx}$(Hu) = 7C33D221368785B7488D8F1D8FD8434EC0C15FA89DAD5C65E3523F57DDD201FF493720A6AC2BA4E930CFCDBB057D1612E337106EE7E177E75FE3D5AC0D436BB0

```
Choose your option(1=Key generator, 2= Hash, 3 = Encrypt or Dencrypt):
3ChaCha20 ...136

Choose your option(1=encrypt, 2=decrypt):
1Input 32 bytes
=> Input key
*ôIÆþÃÓm¨ã65j,8b¿á'‰«aÖÐåÃ˜'ýO
Max of plain text and cipher text is 64 byte:
Input in Hex format:
=> Input plain text for encryption:
 ÃûUÑ¯ð["ýQiXwGpå»òŠúF› ó|KúHA9éÝÇ@ó®´ o+Ò^$ñ%/`&£k*|]ç¬=ChaCha20 256-bit
... ENCRYPT:
PLAIN TEXT:
1A201BC4FB55D1AFF05B94FD51EF5877477007E2BBF28AFA461F9B20F37C4BFA48410E39E9
1DDDC740F3AEB48D6F2BD25E24F1252F6026A36B2A7C5DE7AC3D07
CIPHER TEXT:
7C33D221368785B7488D8F1D8FD8434EC0C15FA89DAD5C65E3523F57DDD201FF493720A6AC
2BA4E930CFCDBB057D1612E337106EE7E177E75FE3D5AC0D436BB0
Passed
```

5) A file contained (IDu,CAu,Ek(Hu),T1) will be sent to server for further calculation (though FTP connection)

```
IDU: 65737038323636
Ekx(Hu) = 7C33D221368785B7488D8F1D8FD8434EC0C15FA89DAD5C65E3523F57DDD201
T1 = 3.20 pm
```

//==========

**Step 1 :** Server received *IDU, CAU, EKX(HU), T1* from user

On the server side, generate a key pair public key *QS* and private key *qS* from ECC with Curve 25519

```
byte[] severRandomBytes = new byte[32] ;
RNGCryptoServiceProvider.Create().GetBytes(severRandomBytes);

byte[] severPrivate = Curve25519.ClampPrivateKey(severRandomBytes);
byte[] severPublic = Curve25519.GetPublicKey(severPrivate);
```

Test:

*qS* in hex: 489e9d12939e79a9a776a9dc62a31c89ad387da86bf6961f5591ecc124799579

*QS* in hex: 7972ebca0f5baa7dfb5076b6e01d48e003f9da35e98f0c243d406fdae2061940

After receiving the session initiation request from user/IoT device in time T2, server checks $|T2 - T1| \leq \Delta T$? If right , sever will retrieve the user's identity IDU and public key QU from CAU.

Test case :

*IDU : 65737038323636*

QU : 2c2745c29076f3d0a6d474b7d69b27154787f7e9ac595c65c841d8a759b1ca4d

Checks retrieved IDU = received IDU? If fails the communication is terminated

Calculates the symmetric shared key K=qS.QU = (KX,KY)

Test case :

K = 2af449c6fec3d36da8e336356a2c3862bfe1049189ab61d6d0e5c41e9891fd4f

Using KX and gets HU

Test case :

HU : 1a201bc4fb55d1aff05b94fd51ef5877477007e2bbf28afa461f9b20f37c4bfa48410e39e91dddc740f3aeb48d6f2bd25e24f1252f6026a36b2a7c5de7ac3d07

Selects a random number *r:*

Test case:

*r:*70591fd9b149f6e6b2cf7a5818ef22cc788c50fc30105609b851a5aa339b1a40

Calculate the server's random point *RS =rS.QS*

Test case :

*RS* = b954e2969758b99dace5b2c26414f2883af55dc59dffd068b3570bed48372851

Caculate DIDU = h (IDU || K || HU)  with SHA512

Test case :

DIDu = 064aab244d8ba35446f207c6f86cf69b7be476cc580b1e100b865297ecdf9fb84307dcf95ae96c952cc7721316525a980eeb7b3b31dd4825d3c5bfee5872075b

Then the concatenated message is encrypted using symmetric key KX  and finally sends the IDS, encrypted message and T2 to IoT device as server challenge.

//==========

**Step 2:**

- User retrieve IDs,Ekx(DIDu||Rs),T3

1)

Verify if  is true if not the communication is terminated

Else Decrypt the server challenge in this case the value will be:

DIDu = 064AAB244D8BA35446F207C6F86CF69B7BE476CC580B1E100B865297ECDF9FB84307DCF95AE96C952CC7721316525A980EEB7B3B31DD4825D3C5BFEE5872075B

Rs = B9 54 E2 96 97 58 B9 9D AC E5 B2 C2 64 14 F2 88 3A F5 5D C5 9D FF D0 68 B3 57 0B ED 48 37 28 51

2)

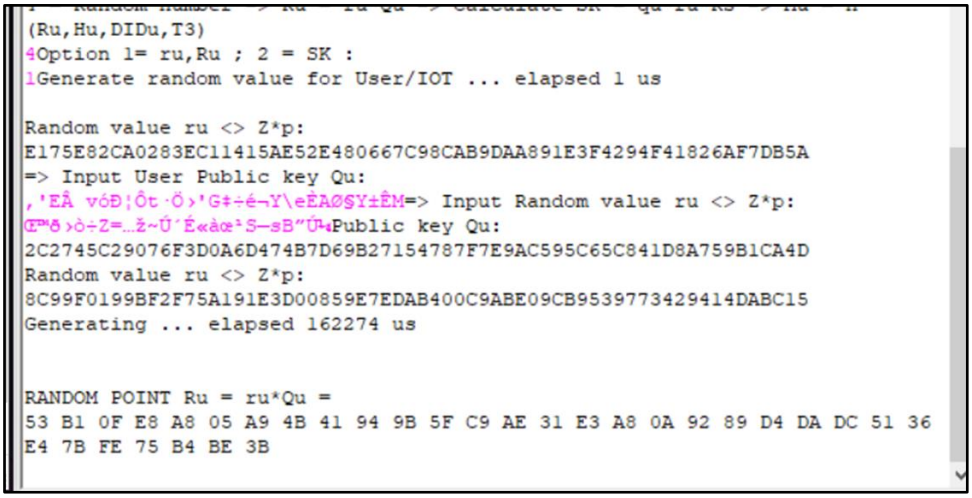
Then calculate user own DIDu value:

```
Choose your option:
1 = Encrypt & Decrypt
2 = SHA512 Hash
3 = DIDu = h(IDu||K||Hu)
4 = Random number -> Ru = ru*Qu -> Calculate SK = qu*ru*Rs -> Mu = h
(Ru,Hu,DIDu,T3)
3Please input (IDu||K||Hu) in hex format:
esp8266*ôIÆþÃÓm¨ã65j,8b¿á'‰«aÖÐåÄ˜'ýO ÄûUÑ¯õ[”ýQïXwGpâ»òŠúF›
ó|KúHA9éÝÇ@ó®´ o+Ò^$ñ%/`&£k*|]ç¬=
DIDu:  ...
Hash value:
064AAB244D8BA35446F207C6F86CF69B7BE476CC580B1E100B865297ECDF9FB84307DCF95A
E96C952CC7721316525A980EEB7B3B31DD4825D3C5BFEE5872075B
Passed
```

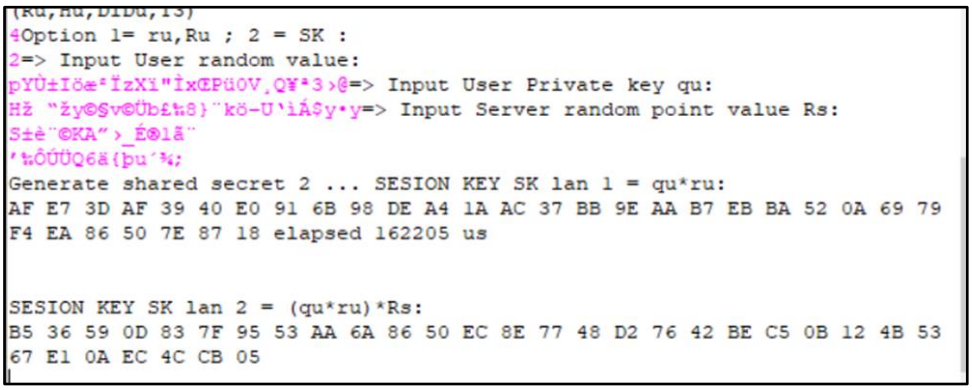
3) User selected:

Random value ru <> Z*p: 8C99F0199BF2F75A191E3D00859E7EDAB400C9ABE09CB9539773429414DABC15

RANDOM POINT Ru = ru*Qu =  53 B1 0F E8 A8 05 A9 4B 41 94 9B 5F C9 AE 31 E3 A8 0A 92 89 D4 DA DC 51 36 E4 7B FE 75 B4 BE 3B




4) Calculate SK = qu*ru*Rs = B5 36 59 0D 83 7F 95 53 AA 6A 86 50 EC 8E 77 48 D2 76 42 BE C5 0B 12 4B 53 67 E1 0A EC 4C CB 05




5) Caculate

Ru||Hu||DiDu||T3 =

53B10FE8A805A94B41949B5FC9AE31E3A80A9289D4DADC5136E47BFE75B4BE3B
1A201BC4FB55D1AFF05B94FD51EF5877477007E2BBF28AFA461F9B20F37C4BFA48410E39E91DDDC740F3
AEB48D6F2BD25E24F1252F6026A36B2A7C5DE7AC3D07

064AAB244D8BA35446F207C6F86CF69B7BE476CC580B1E100B865297ECDF9FB84307DCF95AE96C952CC77
21316525A980EEB7B3B31DD4825D3C5BFEE5872075B

=> Mu = h(Ru||Hu||DiDu||T3) =

b3557c90c4e560b87dff47b640b90f16ba855d28ab227b9dd5c7d60774c030679e193f1a882e1a7c3f74855b9ea0088824
5ee65dfb27b147bb11506841b5021c

=> $E_{SK}(Mu)$ =
64E7D8A42BF834E682DC684B61CE357DFF421E7957B89B88DC5B70DECBA6B4A367CD4C7150C17ED38C0
FF8A2C70905221869299FB2E7F6692981B9673CF68027

//==========

**Step 3:**

The sever received the user challenge

Sever will calculates the session key

*SK = qS.rS .RU = qS.rS.rU.qU.P = qU.rU.rS.qS..P =* (*SKX*,*SKY*), (ii) decrypts the encrypted client challenge and gets *MU* as *DSKX*(*ESKX* (*MU*))= *MU* (iii)

Test case :

SK = b536590d837f9553aa6a8650ec8e7748d27642bec50b124b5367e10aec4ccb05

Calculates *MU /= h(received RU||Stored HU||Stored DIDU||T3)*and (iv) checks *MU /= MU?* If both are equal, the IoT device is authenticated to server.

Test case :

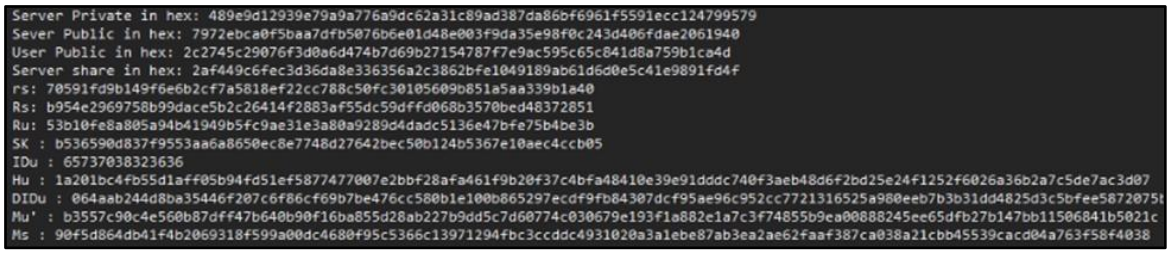
*MU* ‘ = b3557c90c4e560b87dff47b640b90f16ba855d28ab227b9dd5c7d60774c030679e193f1a882e1a7c3f74855b9ea00888245ee65dfb27b147bb11506841b5021c

Now the server calculates *MS =h (RS||MU)*, encrypts *MS* using session key *SKX* and finally sends the encrypted *MS* and the current timestamp *T4* as a server’s response to IoT device.

Test case :

*MS* = 90f5d864db41f4b2069318f599a00dc4680f95c5366c13971294fbc3ccddc4931020a3a1ebe87ab3ea2ae62faaf387ca038a21cbb45539cacd04a763f58f4038

Overview :




Encrypt Ms:

$E_{SK}(M_s)$ = 47477C50345CA0ECF9B03708B8D737AF2DC8D694CAF6F3821B085D1A73BB4057E9F4D0CA33071E1C59519BD6F35A8A603FBDEE09FD957EE45F944E6C88CCC203

//==========

**Step 4:**

Rs||Mu = B954E2969758B99DACE5B2C26414F2883AF55DC59DFFD068B3570BED48372851b3557c90c4e560b87dff47b640b90f16ba855d28ab227b9dd5c7d60774c030679e193f1a882e1a7c3f74855b9ea00888245ee65dfb27b147bb11506841b5021c

M’s = h(Rs||Mu) = 90f5d864db41f4b2069318f599a00dc4680f95c5366c13971294fbc3ccddc4931020a3a1ebe87ab3ea2ae62faaf387ca038a21cbb45539cacd04a763f58f4038

DSK(ESK(Ms)) = 90F5D864DB41F4B2069318F599A00DC4680F95C5366C13971294FBC3CCDDC4931020A3A1EBE87AB3EA2AE62FAAF387CA038A21CBB45539CACD04A763F58F4038


M’s = Ms => Server is authenticated for further message communication is done