Robust and Lightweight Remote User Authentication Mechanism for Next-Generation IoT-Based Smart Home.

Group 11
Divyanshu Jha (S20210010069)
Digvijay Ghule (S20210010082)

Abstract:

Smart home is an IoT concept where physical things such as smart TVs, smart security cameras, smart lights, smart ACs, smart locks, etc have built-in sensors, limited processing ability, and short memory are connected to the Wireless Personal Area Network (WPAN). These smart devices are controlled by users remotely through smartphones with the help of the internet. Mobile users, smart gadgets, a home gateway, and a registration authority (RA) are the typical components of a smart home automation system. The gateway or server serves as a connection point between smart devices and remote users and also acts as the registration authority (RA). The wireless access point serves as a bridge between the smart devices and a home gateway or server. Smart IoT devices have limited resources, including low bandwidth, short memory, and short processing power.

Security and privacy are necessary for the smart home environment. Moreover, data should be exchanged between two parties confidentially and without any change. Asymmetric authentication schemes like RSA and ECC are computationally intensive, making them unsuitable for resource-constrained smart devices. Symmetric schemes like Diffie-Hellman (DH) lack authentication, leaving them vulnerable to man-in-the-middle (MITM) attacks.

Smart devices have short processing power and short memory. Therefore, smart devices demand lightweight and secure authentication protocols. We implemented a lightweight and robust remote user authentication mechanism for IoT-based smart homes which is proposed in the research paper. The scheme consists of the registration phase, login phase, key exchange phase and authentication phase. It meets security requirements against multiple attacks such as password guessing attacks, brute-force attacks, impersonate attacks, replay attacks, forgery attacks, denial of service attacks, MITM attacks, etc.

Implementation:

1) Smart Device Registration Phase:

First, every smart device in the smart home system should be registered. At the registration phase, the Registration Authority (RA) i.e the server assigns a unique ID_{SD} to every smart device. The ID of the smart device is stored in the server's database.

2) User Registration Phase:

All users must be registered at RA. The user selects a unique username as an ID_U, and password PSW_U respectively. The user generates hash-based identity HID_U and hash-based password $HPSW_U$ by using a hash function.

$$HID_U = hash(IDU)$$
$$HPSW_U = hash(ID_U \,//\, PSW_U)$$

The user's secret information is saved to the server's database secretly along with the user's email address and mobile number.

3) Login Phase:

During the login phase, the server compares the hash values of the user's credentials with those stored in its database. If they match, the server proceeds to establish a session key; otherwise, it terminates the session..

In our implemented client-server model, the client (Remote user) enters their userID (ID_U) and password (PSW_U) in the GUI window built using the `tkinter` python module and sends it to the server. This communication between client and server is implemented using the `socket` module in python. The server then calculates the hash values of received userID and password using SHA-256 hash function from the `hashlib` module. Server verifies the user by matching the values stored in its database with the calculated hash values (HID_U, $HPSW_U$). Once the user is verified, the server generates 2 random numbers (N_1, N_2) of size 128 bits. We have used the `secrets` module to generate these random numbers. Values N_1 and N_2 are then sent from the server to the client. This completes the Login phase.

4) Session Key Exchange Phase:

After successful login, the remote user will generate a larger random number N_U . The size of the random number will be the same as N1, N2 i.e. 128 bits. Now, the user computes values Res_U and FR_U as follows:

$$Res_U = (N_U \times N_1) + N_1$$
$$FR_U = (Res_U \times N_2) + N_1 + N_2$$

The user finally sends the final result FR_U to the server. The result FR_U is not a key and if the intruder captures it then he can't retrieve the actual key until he knows both secret numbers N_1 and N_2 respectively. When the server receives the result FR_U sent by the remote user then the server extracts the number N_U by using both numbers N_1 and N_2 respectively.

$$Res_S = FR_U - (N_1 + N_2)$$
$$N_U = (Res_S / (N_1 \times N_2)) - 1$$

Similarly, a large random number N_S will be generated on the server side. Value FR_S will be sent to the user by following the same procedure as followed on the client side. At last, the server and the user compute bitwise XOR of N_U with N_S which were received on both sides, calculate mod with M, and get the final symmetric session key K_S on both sides secretly. M is a variable that determines the size of the key (128 bits).

$$K_S = (N_U \oplus N_S) \bmod M$$

The same key K_S has been exchanged between the server and the recognized remote user. On every newly established connection between the remote user and the server, the session key will be changed. The session key is used for authentication.

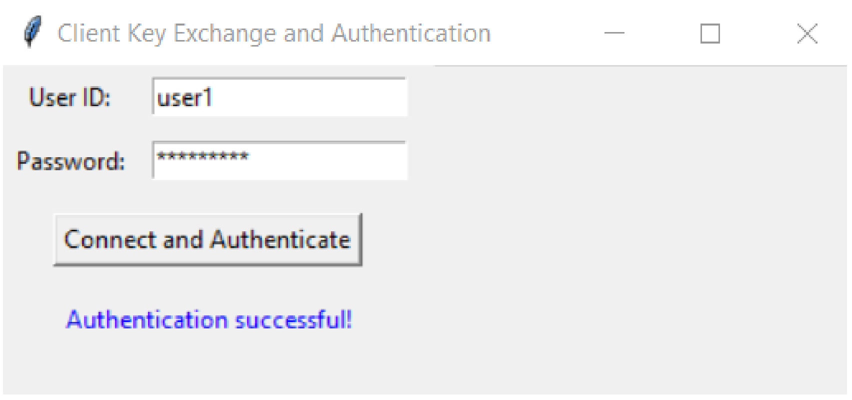
5) Authentication Phase:

We have used HMAC(Hash-based Message Authentication Code) using the `hashlib` module to implement the authentication process. HMAC is a specific type of fixed-length message authentication code that is generated by a hashing algorithm and a secret key. In this scheme, the IPv6 address of the host concatenates with the randomly generated number by the host, and the session key to generate a hash value of size 256 bits through the SHA-256 algorithm as follows:

$$HMAC_U = SHA{-}256\ (IPAddress_U\ //\ \ N_U, K_S)$$
$$HMAC_S = SHA{-}256\ (IPAddress_S\ //\ \ N_S, K_S)$$

The HMAC calculated values are sent to each other on both sides. HMAC values are re-calculated on both sides for cross-checking and verification with the received value. If the calculated HMAC and received HMAC are verified, then the authentication process is completed on both sides. After a successful authentication, the server grants access to the user to control the smart devices. If the authentication process fails on any side, then the connection is terminated immediately on both sides and declared an adversary attack.

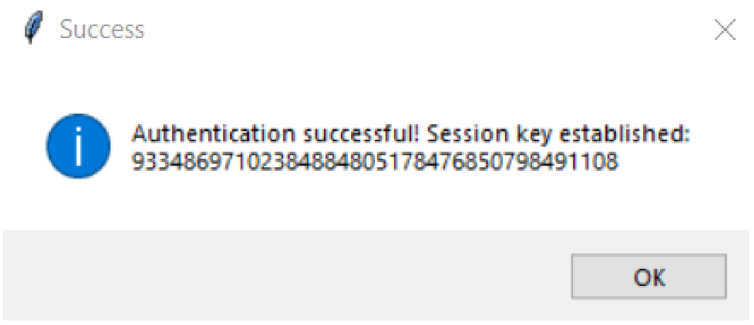
Result/Observation:


Client Key Exchange and Authentication

User ID: user1

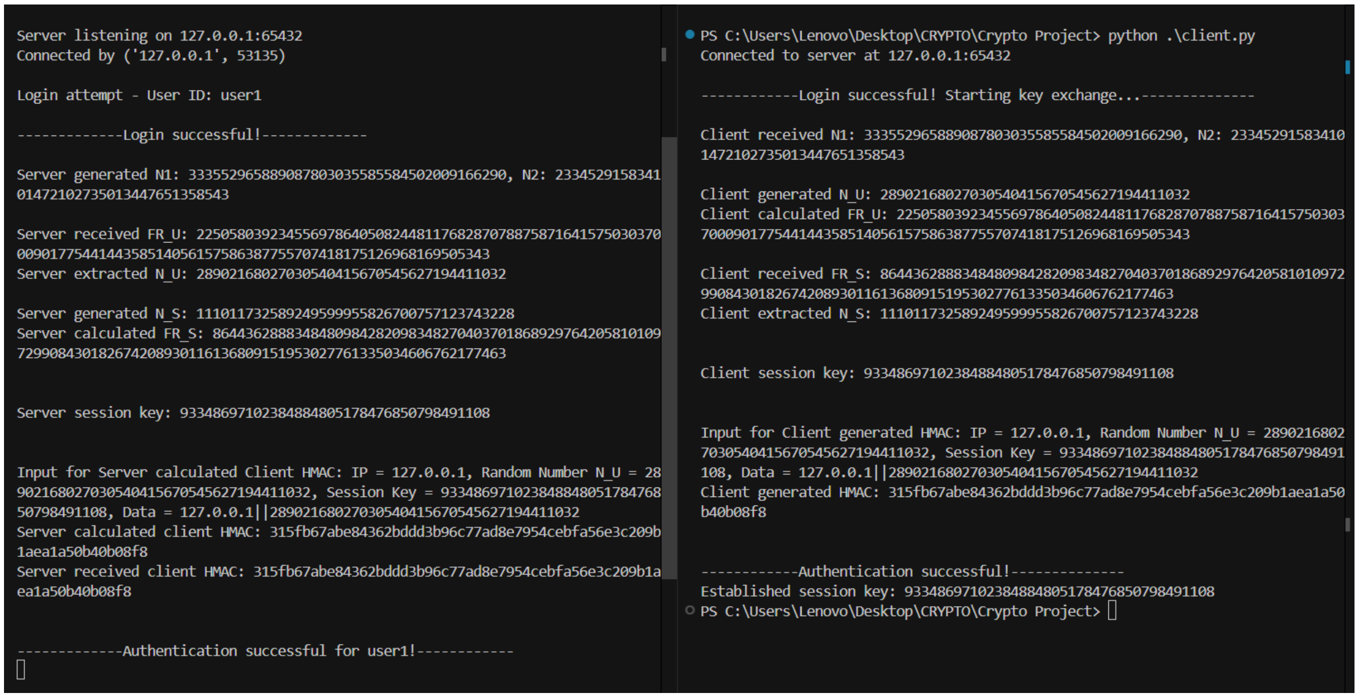
Password: *********

Authentication successful!



Success

Authentication successful! Session key established:
93348697102384884805178476850798491108



```
Server listening on 127.0.0.1:65432
Connected by ('127.0.0.1', 53135)

Login attempt - User ID: user1

------------Login successful!------------

Server generated N1: 333552965889087803035585584502009166290, N2: 2334529158341
01472102735013447651358543

Server received FR_U: 225058039234556978640508244811768287078875871641575030370
0090177544144358514056157586387755707418175126968169505343
Server extracted N_U: 28902168027030540415670545627194411032

Server generated N_S: 111011732589249599955826700757123743228
Server calculated FR_S: 8644362888348480984282098348270403701868929764205810109
729908430182674208930116136809151953027761335034606762177463


Server session key: 93348697102384884805178476850798491108


Input for Server calculated Client HMAC: IP = 127.0.0.1, Random Number N_U = 28
902168027030540415670545627194411032, Session Key = 933486971023848848051784768
50798491108, Data = 127.0.0.1||28902168027030540415670545627194411032
Server calculated client HMAC: 315fb67abe84362bddd3b96c77ad8e7954cebfa56e3c209b
1aea1a50b40b08f8
Server received client HMAC: 315fb67abe84362bddd3b96c77ad8e7954cebfa56e3c209b1a
ea1a50b40b08f8


------------Authentication successful for user1!-----------
```

```
PS C:\Users\Lenovo\Desktop\CRYPTO\Crypto Project> python .\client.py
Connected to server at 127.0.0.1:65432

-----------Login successful! Starting key exchange...-------------

Client received N1: 333552965889087803035585584502009166290, N2: 23345291583410
1472102735013447651358543

Client generated N_U: 28902168027030540415670545627194411032
Client calculated FR_U: 2250580392345569786405082448117682870788758716415750303
700090177544144358514056157586387755707418175126968169505343

Client received FR_S: 864436288834848098428209834827040370186892976420581010972
9908430182674208930116136809151953027761335034606762177463
Client extracted N_S: 111011732589249599955826700757123743228


Client session key: 93348697102384884805178476850798491108


Input for Client generated HMAC: IP = 127.0.0.1, Random Number N_U = 2890216802
7030540415670545627194411032, Session Key = 93348697102384884805178476850798491
108, Data = 127.0.0.1||28902168027030540415670545627194411032
Client generated HMAC: 315fb67abe84362bddd3b96c77ad8e7954cebfa56e3c209b1aea1a50
b40b08f8


-----------Authentication successful!-------------
Established session key: 93348697102384884805178476850798491108
PS C:\Users\Lenovo\Desktop\CRYPTO\Crypto Project> 
```

- We implemented the proposed robust and lightweight authentication scheme by using a pre-shared symmetric session key for the next-generation of IoT-based smart homes.
- We implemented the authentication scheme on a client-server network model with Python socket programming. The proposed scheme is lightweight in terms of computation and communication costs.
- The proposed scheme meets security requirements against multiple attacks such as password guessing attacks, brute-force attacks, impersonate attacks, replay attacks, forgery attacks, denial of service attacks, MITM attacks, perfect forward secrecy, etc.

References:

Ashraf Z., et al. "Robust and Lightweight Remote User Authentication Mechanism for Next-Generation IoT-Based Smart Home." IEEE Access, vol. 11, 2023, pp. 137899-137910.