

Background

The IEEE 802.15.6 standard is the latest globally accepted standard for Wireless Body Area Network (WBAN) and it aims to provide confidentiality and integrity. As the name suggests, WBAN is a wireless network of devices that may be wearable or embedded within an individual's body. These devices would be considered as nodes and hubs within the network. The IEEE standard provides the required methods for secure communication between these nodes. In this report, the IEEE standard is realized and additions that would improve the protocol's security are discussed and analyzed.

Protocol Description

The provided example files `wban1coordinator.py` and `wban1sensor.py` demonstrate an encrypted yet unauthenticated association between the coordinator and the sensor. Due to the lack of authentication in this communication, the two parties are unable to ensure the correctness of the received data after the exchange of their public keys. Therefore, the protocol is vulnerable to man-in-the-middle attacks, where a third party could infiltrate the connection and gain access to the two parties' public keys.

This problem could be solved by implementing a password authenticated association between the coordinator and the sensor. In addition, it is believed that with the addition of acknowledgement messages (ACKs), both parties would be enabled to ensure that the other party have securely received their message. That is, after a party receives a messages, it analyzes the message and checks the correctness of the given parameters. Accordingly, it would send an ACK to the other party to inform them that they have securely received this message. The pseudocode for this protocol is provided below:

- **Initialization**

Parties A and B share group parameters (such as IP, port, and addresses) and password PW.

- **Key Exchange**

- 1) Party A generates private key SK_A and public key PK_A . Accordingly, it selects a random Nonce N_A , computes the password scrambled public key PK_{APS} as $PK_{APS} = PK_A - PW$, sends message $M1 = (Address_A, Address_B, N_A, PK_{APS})$ to B, and waits for B's acknowledgement (ACK).
- 2) Party B receives, verifies and acknowledges (ACKs) M1 and generates private key SK_B and public key PK_B . Consequently, it selects a random Nonce N_B , sends message $M2 = (Address_B, Address_A, N_B, PK_B)$ to A, and waits for A's ACK.
- 3) Party A receives, verifies and ACKs M2. Subsequently, it calculates the shared secret key $dhkey$ as $dhkey = PK_B \cdot SK_A$.
- 4) Party B receives A's ACK. Consequently, it recovers A's public key PK_A by $PK_A = PK_{APS} + PW$, computes shared secret key as $dhkey = PK_A \cdot SK_B$, calculates mac_B as $H(dhkey, Address_A, Address_B, N_A, N_B)$, and sends message $M3 = (Address_A, Address_B, N_B, PK_B, mac_B)$ to A, and waits for A's ACK.
- 5) Party A ACKs M3. Then, it verifies mac_B , computes mac_A as $H(dhkey, Address_B, Address_A, N_B, N_A)$, sends message $M4 = (Address_B, Address_A, N_A, PK_A, mac_A)$ to B and waits for B's ACK.
- 6) Party B ACKs M4 and verifies mac_A .

Experimental Platforms

In this assignment, the platform for the coordinator was a desktop computer running 64-bit Windows10, with 32GB of installed RAM and a 2.4GHZ CPU. Visual Studio Code IDE alongside the GitBash terminal were used for developing and testing the code on this platform. Moreover, the sensor was tested on a virtual machine with 10GB of RAM and 1.2GHZ CPU with Ubuntu18.04 as its' operating system. The Bash terminal within Visual Studio Code was used to test the code on this platform likewise.

Experimental Results

The results of multiple runs of the program are provided respectively below. As illustrated in the figures, A refers to the coordinator while B indicates the sensor.

```

Sahand@Sahand MINGW64 ~/Desktop/Lab 1 codes [Python 3]
$ python ieee802.15.6/coordinator.py
Listen to connections from initiators...
Connected to sensor via ('192.168.0.14', 59546)
M2 Acknowledged! Continuing the protocol
M3 Acknowledged! Continuing the protocol
macb is valid
Shared secret key => (8830297144614070224386405785968395753104881990233257595654288936558797571172, 22939508791680305789992109062846276111853262643638701307381318019834869206283)

Runtime: 0.145000 seconds

samsepiol@theviper:~/Desktop/Lab 1 codes [Python 3]$ python3 ieee802.15.6/sensor.py
Initializing socket connection to coordinator B...
Socket connected to coordinator B
M1 Acknowledged! Continuing the protocol
macb is valid
M4 Acknowledged! Continuing the protocol
Shared secret key => (8830297144614070224386405785968395753104881990233257595654288936558797571172, 22939508791680305789992109062846276111853262643638701307381318019834869206283)

Runtime: 0.142895 seconds

```

Figure 1: Protocol's first test run

```

Sahand@Sahand MINGW64 ~/Desktop/Lab 1 codes [Python 3]
$ python ieee802.15.6/coordinator.py
Listen to connections from initiators...
Connected to sensor via ('192.168.0.14', 59580)
M2 Acknowledged! Continuing the protocol
M3 Acknowledged! Continuing the protocol
macb is valid
Shared secret key => (78261841348201064229615738756124090315046149334374642347435497366818330150922, 55892218641428675145208072173267880965595572963888139448942103942696363981712)

Runtime: 0.151997 seconds

samsepiol@theviper:~/Desktop/Lab 1 codes [Python 3]$ python3 ieee802.15.6/sensor.py
Initializing socket connection to coordinator B...
Socket connected to coordinator B
M1 Acknowledged! Continuing the protocol
macb is valid
M4 Acknowledged! Continuing the protocol
Shared secret key => (78261841348201064229615738756124090315046149334374642347435497366818330150922, 55892218641428675145208072173267880965595572963888139448942103942696363981712)

Runtime: 0.152046 seconds

```

Figure 2: Protocol's second test run



```

Sahand@Sahand MINGW64 ~/Desktop/Lab 1 codes [Python 3]
$ python ieee802.15.6/coordinator.py
Listen to connections from initiators...
Connected to sensor via ('192.168.0.14', 59600)
M2 Acknowledged! Continuing the protocol
M3 Acknowledged! Continuing the protocol
maca is valid
Shared secret key => (55811973473324435115344713178961993765703683368263010039014670963395143625408, 11277987780573104820499589789698688055726484533621519232747364434614044190590)

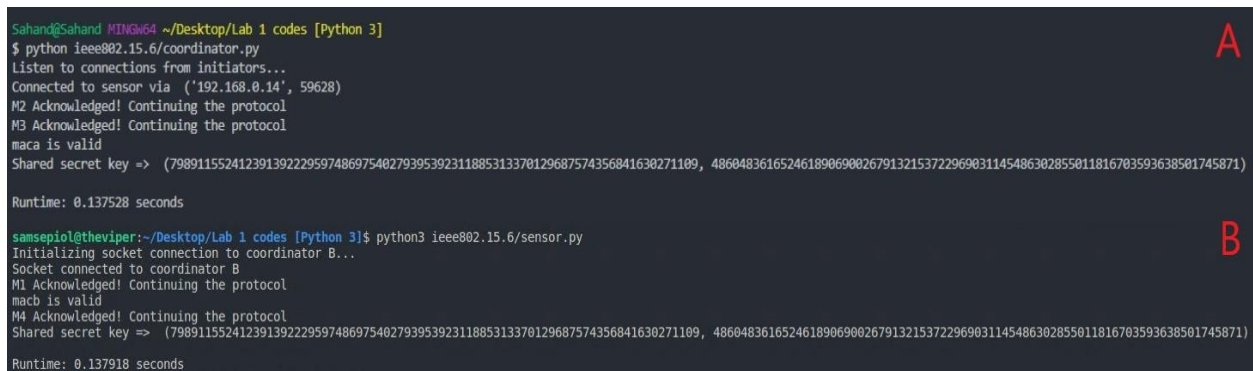
Runtime: 0.136003 seconds

samsepiol@theviper:~/Desktop/Lab 1 codes [Python 3]$ python3 ieee802.15.6/sensor.py
Initializing socket connection to coordinator B...
Socket connected to coordinator B
M1 Acknowledged! Continuing the protocol
macb is valid
M4 Acknowledged! Continuing the protocol
Shared secret key => (55811973473324435115344713178961993765703683368263010039014670963395143625408, 11277987780573104820499589789698688055726484533621519232747364434614044190590)

Runtime: 0.136617 seconds

```

Figure 3: Protocol's third test run



```

Sahand@Sahand MINGW64 ~/Desktop/Lab 1 codes [Python 3]
$ python ieee802.15.6/coordinator.py
Listen to connections from initiators...
Connected to sensor via ('192.168.0.14', 59628)
M2 Acknowledged! Continuing the protocol
M3 Acknowledged! Continuing the protocol
maca is valid
Shared secret key => (79891155241239139222959748697540279395392311885313370129687574356841630271109, 48604836165246189069002679132153722969031145486302855011816703593638501745871)

Runtime: 0.137528 seconds

samsepiol@theviper:~/Desktop/Lab 1 codes [Python 3]$ python3 ieee802.15.6/sensor.py
Initializing socket connection to coordinator B...
Socket connected to coordinator B
M1 Acknowledged! Continuing the protocol
macb is valid
M4 Acknowledged! Continuing the protocol
Shared secret key => (79891155241239139222959748697540279395392311885313370129687574356841630271109, 48604836165246189069002679132153722969031145486302855011816703593638501745871)

Runtime: 0.137918 seconds

```

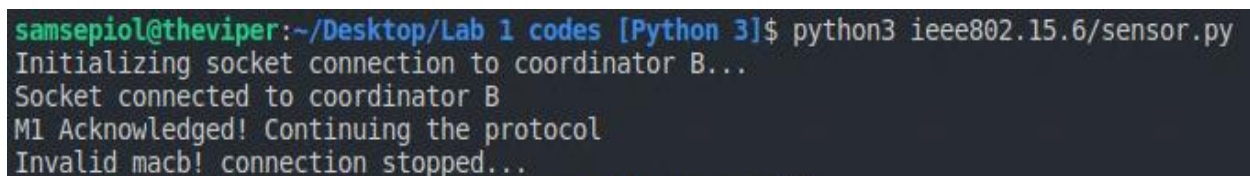
Figure 4: Protocol's fourth test run

Based on the above figures, it can be derived that the two parties were able to successfully establish a secure connection and share a secret key. The runtime for each platform is provided respectively below.

Attempt #	Coordinator	Sensor
1	0.145000 s	0.142895 s
2	0.152000 s	0.152046 s
3	0.136003 s	0.136617 s
4	0.137528 s	0.137918 s
Average	0.142633 s	0.142369 s

Table 1: Program runtime chart

Moreover, the following test demonstrates the case in which the two parties have different passwords. As shown in the figure, the protocol detects the unauthenticity and stops as expected.



```

samsepiol@theviper:~/Desktop/Lab 1 codes [Python 3]$ python3 ieee802.15.6/sensor.py
Initializing socket connection to coordinator B...
Socket connected to coordinator B
M1 Acknowledged! Continuing the protocol
Invalid macb! connection stopped...

```

Figure 5: Man-in-the-middle test