Georg-August-Universität Göttingen

Faculty of Mathematics and Computer Science

Institute of Computer Science

Telematics Group

Sensorlab

Lab5–Encrypted Communication (Symmetric)

Chencheng Liang – University of Goettingen – 11603960

Introduction

In this lab, through check the code in crypto to understand how to encrypt and decrypt the plaintext.

In assignment 5.1, By adapting the python code from lab 3, I simulate a application RadioCountToLeds, to finish that work, in which the nodes will send and receive messages to each other, but the encryption is finished only in senders.

In assignment 5.2, By using the encryption mode, and check the code inside the files such as CTRModeM, BlockCipher, etc. I can put the right parameters to the functions, so I can use them for encryption.

In question 5.2, there are lot of theories should to be known, then I can answer these questions, I need to understand every concept and compare them, and consider them how to connect them with WSN.

In this lab I get many interesting results, some of them are hard to explain.

All the details will been found in questions and assignments.

Question 5. 1

There is no differences of implement of BlockCipherInfo in XTEAM.nc and AES128M.nc.

In some sense, it is still useful, because decryption can be done in many ways, not only by decryption function. Due to block cipher is a symmetric encryption method, so if we know the algorithm, and intercept the key every one can decrypt that. However, for very sophisticate block cipher, the decryption function is preferred and convenient.

Assignment 5. 1

I adapt TOSSIM example from lab3 to this assignment. Node send and receive packages, and use dbg() to print information in their channels. The all encryption processes are done in the sender.

In RadioCountToLedsAppC.nc, add components CTRModeM and AES128M, and wire them(App.BlockCipher -> AES128M , App.BlockCipherInfo -> AES128M and App.Mode -> CTRModeM;).

In RadioCountToLedsC.nc, declare some variables to save cipher text(uint8_t cipherText[16]="KAMI_SAME") , encrypted message(uint8_t encryptedText[16]), decrypted message(uint8_t encryptedText[16]), key, and other useful information.

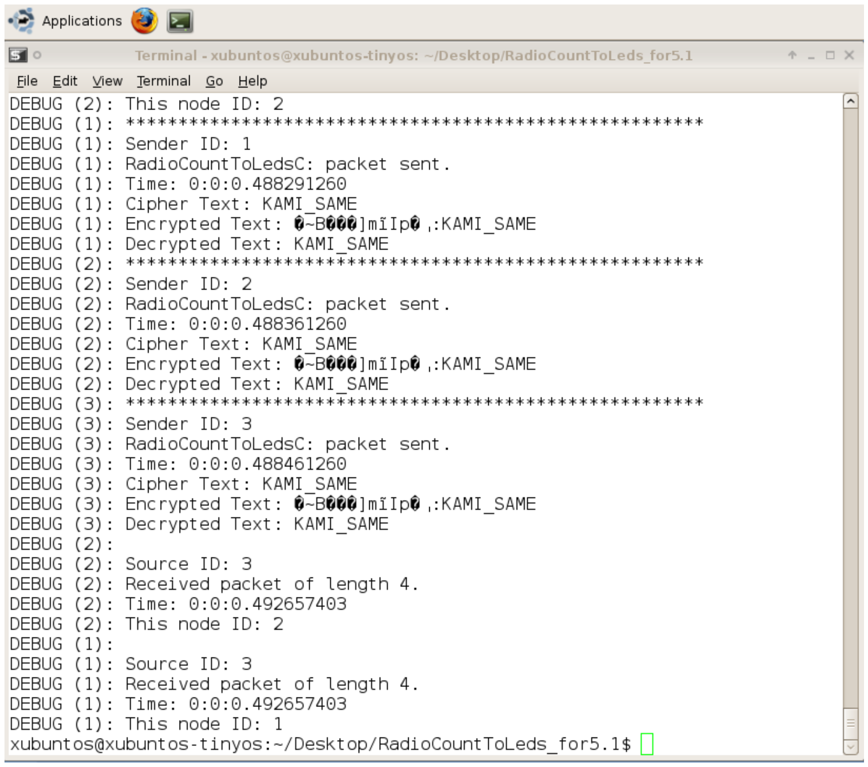
Use " BlockSize = call BlockCipherInfo.getPreferredBlockSize(); " to get block size, and " KeyLength = call BlockCipherInfo.getMaxKeyLength()" to get key length, which are needed in initializing block cipher.

The by calling BlockCipher.init(), to initialize the block cipher, and call BlockCipher.encrypt() to encrypt the cipher text. Next call BlockCipher.decrypt() to decrypt the encrypted text.

Then print them on the screen by dbg() in "Enc" channel, which channel also need to be add in the unSim.py code.

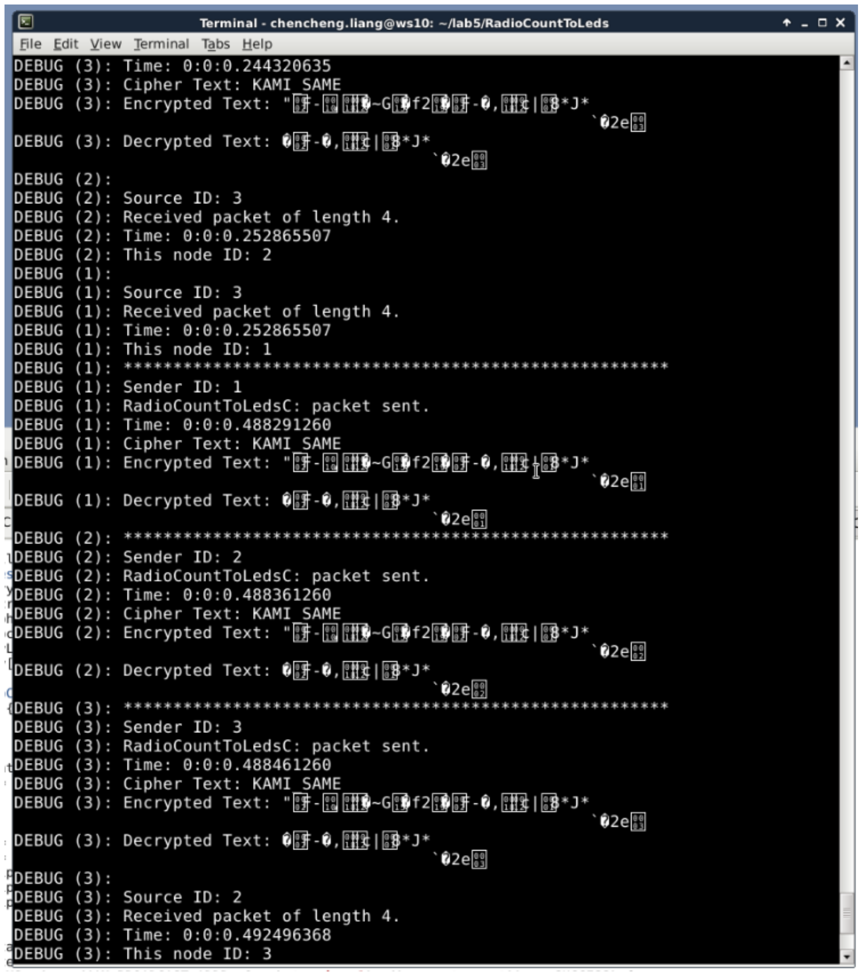
After the code finished, run TOSSIM by command "python unSim.py", then I get the result:

This is at home:



```
DEBUG (2): This node ID: 2
DEBUG (1): ******************************************************
DEBUG (1): Sender ID: 1
DEBUG (1): RadioCountToLedsC: packet sent.
DEBUG (1): Time: 0:0:0.488291260
DEBUG (1): Cipher Text: KAMI_SAME
DEBUG (1): Encrypted Text: �~B���]mĩIp� ,:KAMI_SAME
DEBUG (1): Decrypted Text: KAMI_SAME
DEBUG (2): ******************************************************
DEBUG (2): Sender ID: 2
DEBUG (2): RadioCountToLedsC: packet sent.
DEBUG (2): Time: 0:0:0.488361260
DEBUG (2): Cipher Text: KAMI_SAME
DEBUG (2): Encrypted Text: �~B���]mĩIp� ,:KAMI_SAME
DEBUG (2): Decrypted Text: KAMI_SAME
DEBUG (3): ******************************************************
DEBUG (3): Sender ID: 3
DEBUG (3): RadioCountToLedsC: packet sent.
DEBUG (3): Time: 0:0:0.488461260
DEBUG (3): Cipher Text: KAMI_SAME
DEBUG (3): Encrypted Text: �~B���]mĩIp� ,:KAMI_SAME
DEBUG (3): Decrypted Text: KAMI_SAME
DEBUG (2):
DEBUG (2): Source ID: 3
DEBUG (2): Received packet of length 4.
DEBUG (2): Time: 0:0:0.492657403
DEBUG (2): This node ID: 2
DEBUG (1):
DEBUG (1): Source ID: 3
DEBUG (1): Received packet of length 4.
DEBUG (1): Time: 0:0:0.492657403
DEBUG (1): This node ID: 1
xubuntos@xubuntos-tinyos:~/Desktop/RadioCountToLeds_for5.1$
```

This is at lab's computer:



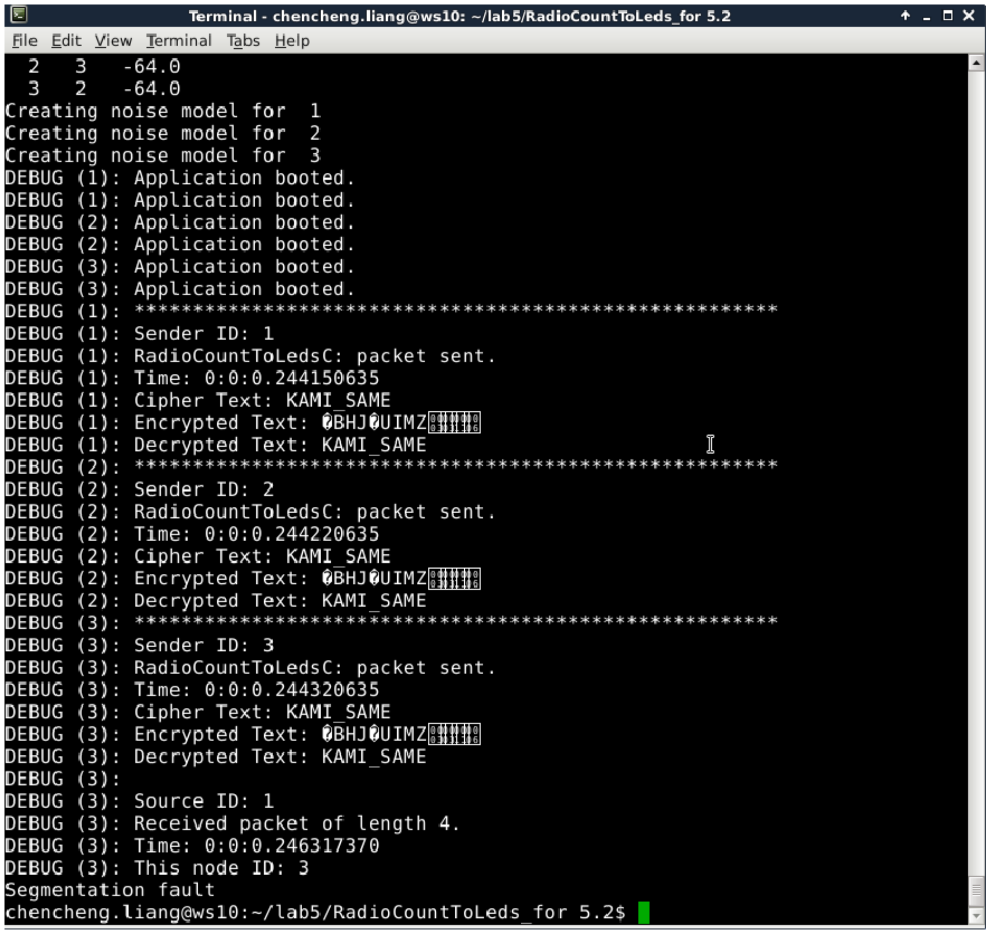
```
DEBUG (3): Time: 0:0:0.244320635
DEBUG (3): Cipher Text: KAMI_SAME
DEBUG (3): Encrypted Text: "�f-� ��~G�f2��f-�,��c|�8*J*
                                                     `�2e�

DEBUG (3): Decrypted Text: ��f-�,��c|�8*J*
                                       `�2e�

DEBUG (2):
DEBUG (2): Source ID: 3
DEBUG (2): Received packet of length 4.
DEBUG (2): Time: 0:0:0.252865507
DEBUG (2): This node ID: 2
DEBUG (1):
DEBUG (1): Source ID: 3
DEBUG (1): Received packet of length 4.
DEBUG (1): Time: 0:0:0.252865507
DEBUG (1): This node ID: 1
DEBUG (1): ***********************************************************
DEBUG (1): Sender ID: 1
DEBUG (1): RadioCountToLedsC: packet sent.
DEBUG (1): Time: 0:0:0.488291260
DEBUG (1): Cipher Text: KAMI_SAME
DEBUG (1): Encrypted Text: "�f-� ��~G�f2��f-�,��c|�8*J*
                                                     `�2e�

DEBUG (1): Decrypted Text: ��f-�,��c|�8*J*
                                       `�2e�
DEBUG (2): ***********************************************************
DEBUG (2): Sender ID: 2
DEBUG (2): RadioCountToLedsC: packet sent.
DEBUG (2): Time: 0:0:0.488361260
DEBUG (2): Cipher Text: KAMI_SAME
DEBUG (2): Encrypted Text: "�f-� ��~G�f2��f-�,��c|�8*J*
                                                     `�2e�

DEBUG (2): Decrypted Text: ��f-�,��c|�8*J*
                                       `�2e�
DEBUG (3): ***********************************************************
DEBUG (3): Sender ID: 3
DEBUG (3): RadioCountToLedsC: packet sent.
DEBUG (3): Time: 0:0:0.488461260
DEBUG (3): Cipher Text: KAMI_SAME
DEBUG (3): Encrypted Text: "�f-� ��~G�f2��f-�,��c|�8*J*
                                                     `�2e�

DEBUG (3): Decrypted Text: ��f-�,��c|�8*J*
                                       `�2e�
DEBUG (3):
DEBUG (3): Source ID: 2
DEBUG (3): Received packet of length 4.
DEBUG (3): Time: 0:0:0.492496368
DEBUG (3): This node ID: 3
```

That is so strange! Because I used the same code, in lab the decryption function did not work at all, in home the decryption function worked, but the encryption function did not work well! It shows my cipher text after encryption.

It is so interesting! However, due to the submit dead line I do not have time to figure out why.

Assignment 5. 2

It is same like last assignment, but use BlockCipherMode interface instead of using BlockCipher directly. The detail is in the code, this time there are also interesting thing happened, this is the result at lab:

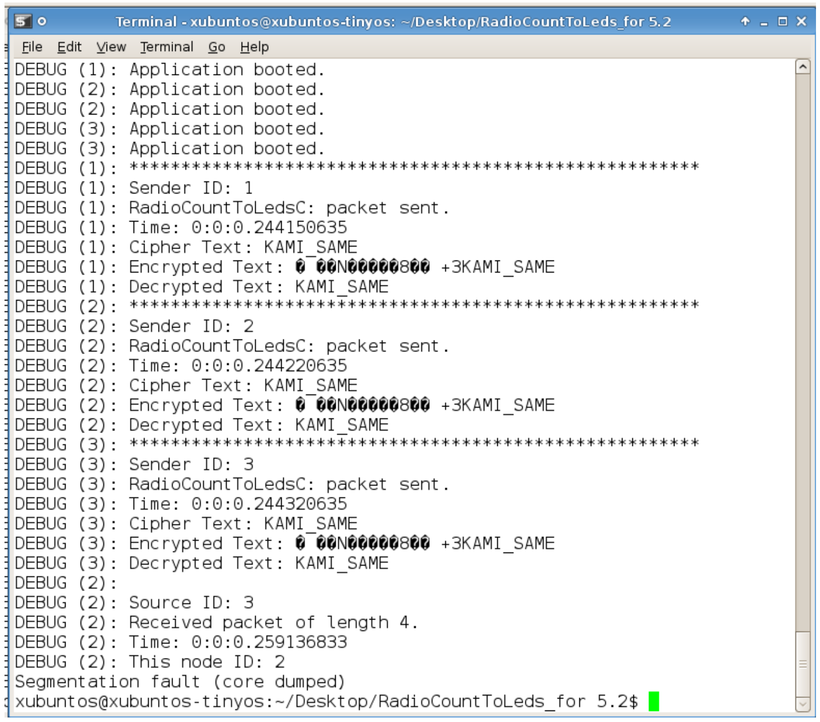


Terminal - chencheng.liang@ws10: ~/lab5/RadioCountToLeds_for 5.2
File Edit View Terminal Tabs Help

```
  2   3   -64.0
  3   2   -64.0
Creating noise model for  1
Creating noise model for  2
Creating noise model for  3
DEBUG (1): Application booted.
DEBUG (1): Application booted.
DEBUG (2): Application booted.
DEBUG (2): Application booted.
DEBUG (3): Application booted.
DEBUG (3): Application booted.
DEBUG (1): *******************************************************
DEBUG (1): Sender ID: 1
DEBUG (1): RadioCountToLedsC: packet sent.
DEBUG (1): Time: 0:0:0.244150635
DEBUG (1): Cipher Text: KAMI_SAME
DEBUG (1): Encrypted Text: �BHJ�UIMZ����
DEBUG (1): Decrypted Text: KAMI_SAME
DEBUG (2): *******************************************************
DEBUG (2): Sender ID: 2
DEBUG (2): RadioCountToLedsC: packet sent.
DEBUG (2): Time: 0:0:0.244220635
DEBUG (2): Cipher Text: KAMI_SAME
DEBUG (2): Encrypted Text: �BHJ�UIMZ����
DEBUG (2): Decrypted Text: KAMI_SAME
DEBUG (3): *******************************************************
DEBUG (3): Sender ID: 3
DEBUG (3): RadioCountToLedsC: packet sent.
DEBUG (3): Time: 0:0:0.244320635
DEBUG (3): Cipher Text: KAMI_SAME
DEBUG (3): Encrypted Text: �BHJ�UIMZ����
DEBUG (3): Decrypted Text: KAMI_SAME
DEBUG (3):
DEBUG (3): Source ID: 1
DEBUG (3): Received packet of length 4.
DEBUG (3): Time: 0:0:0.246317370
DEBUG (3): This node ID: 3
Segmentation fault
chencheng.liang@ws10:~/lab5/RadioCountToLeds_for 5.2$
```

The encryption functions work very well, the encrypted text cannot be seen, and decrypted text is correct.

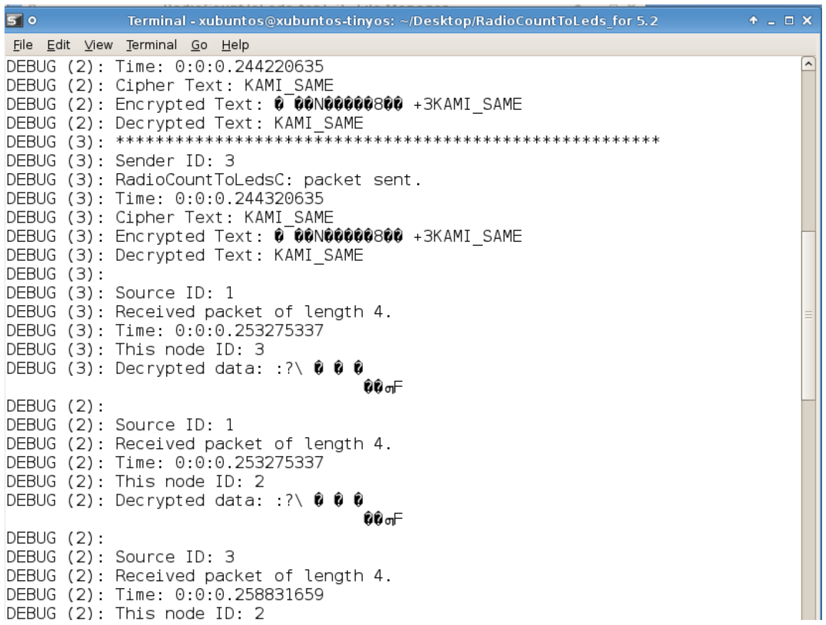
But when I get this code to home, the result is this:


Terminal - xubuntos@xubuntos-tinyos: ~/Desktop/RadioCountToLeds_for 5.2
File Edit View Terminal Go Help

```
DEBUG (1): Application booted.
DEBUG (2): Application booted.
DEBUG (2): Application booted.
DEBUG (3): Application booted.
DEBUG (3): Application booted.
DEBUG (1): ******************************************************
DEBUG (1): Sender ID: 1
DEBUG (1): RadioCountToLedsC: packet sent.
DEBUG (1): Time: 0:0:0.244150635
DEBUG (1): Cipher Text: KAMI_SAME
DEBUG (1): Encrypted Text: � ��N�����8�� +3KAMI_SAME
DEBUG (1): Decrypted Text: KAMI_SAME
DEBUG (2): ******************************************************
DEBUG (2): Sender ID: 2
DEBUG (2): RadioCountToLedsC: packet sent.
DEBUG (2): Time: 0:0:0.244220635
DEBUG (2): Cipher Text: KAMI_SAME
DEBUG (2): Encrypted Text: � ��N�����8�� +3KAMI_SAME
DEBUG (2): Decrypted Text: KAMI_SAME
DEBUG (3): ******************************************************
DEBUG (3): Sender ID: 3
DEBUG (3): RadioCountToLedsC: packet sent.
DEBUG (3): Time: 0:0:0.244320635
DEBUG (3): Cipher Text: KAMI_SAME
DEBUG (3): Encrypted Text: � ��N�����8�� +3KAMI_SAME
DEBUG (3): Decrypted Text: KAMI_SAME
DEBUG (2):
DEBUG (2): Source ID: 3
DEBUG (2): Received packet of length 4.
DEBUG (2): Time: 0:0:0.259136833
DEBUG (2): This node ID: 2
Segmentation fault (core dumped)
xubuntos@xubuntos-tinyos:~/Desktop/RadioCountToLeds_for 5.2$
```

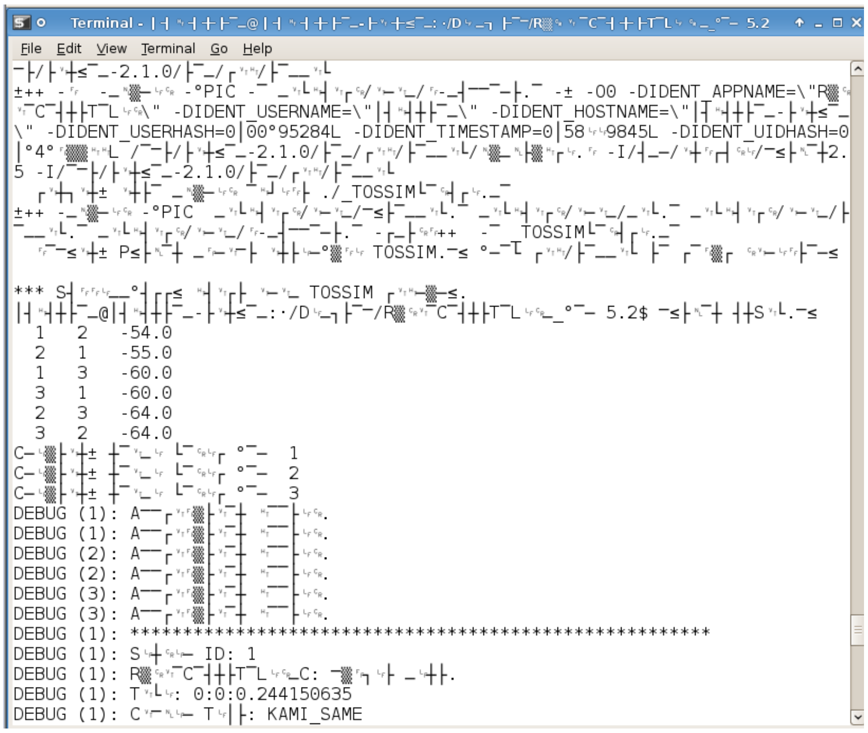
The encryption still exposed my cipher text!

If we use counter mode, there are problems occurs, because the receiver cannot get the correct counter due to package lost, then the receiver cannot decrypt the text properly like this:


Terminal - xubuntos@xubuntos-tinyos: ~/Desktop/RadioCountToLeds_for 5.2
File Edit View Terminal Go Help

```
DEBUG (2): Time: 0:0:0.244220635
DEBUG (2): Cipher Text: KAMI_SAME
DEBUG (2): Encrypted Text: � ��N�����8�� +3KAMI_SAME
DEBUG (2): Decrypted Text: KAMI_SAME
DEBUG (3): ******************************************************
DEBUG (3): Sender ID: 3
DEBUG (3): RadioCountToLedsC: packet sent.
DEBUG (3): Time: 0:0:0.244320635
DEBUG (3): Cipher Text: KAMI_SAME
DEBUG (3): Encrypted Text: � ��N�����8�� +3KAMI_SAME
DEBUG (3): Decrypted Text: KAMI_SAME
DEBUG (3):
DEBUG (3): Source ID: 1
DEBUG (3): Received packet of length 4.
DEBUG (3): Time: 0:0:0.253275337
DEBUG (3): This node ID: 3
DEBUG (3): Decrypted data: :?\ � � �
                                       ��σF
DEBUG (2):
DEBUG (2): Source ID: 1
DEBUG (2): Received packet of length 4.
DEBUG (2): Time: 0:0:0.253275337
DEBUG (2): This node ID: 2
DEBUG (2): Decrypted data: :?\ � � �
                                       ��σF
DEBUG (2):
DEBUG (2): Source ID: 3
DEBUG (2): Received packet of length 4.
DEBUG (2): Time: 0:0:0.258831659
DEBUG (2): This node ID: 2
```

The most interesting thing is when I use XTEAM, my whole terminal become garbled like this:



```
─├/├ ┼≤¯_-2.1.0/├¯_/┌ /├¯__ └
±++ -  -_ ▒─  -°PIC -¯ _ └ ┤ ┌ / ─ _/ -_┤──¯─├.¯ -± -O0 -DIDENT_APPNAME=\"R▒
 ¯C¯┤┼├T¯L \" -DIDENT_USERNAME=\"|┤ ┤┼├¯_\" -DIDENT_HOSTNAME=\"|┤ ┤┼├¯_-├ ┼≤¯_
\" -DIDENT_USERHASH=0|00°95284L -DIDENT_TIMESTAMP=0|58 9845L -DIDENT_UIDHASH=0
|°4° ▒ L /¯─├/├ ┼≤¯_-2.1.0/├¯_/┌ /├¯__ └/ ▒_ ├▒ ┌ .  -I/┤_─/ ┼ ┌┤ /¯≤├ ¯┼2.
5 -I/¯─├/├ ┼≤¯_-2.1.0/├¯_/┌ /├¯__ └
  ┌ ┼┐ ┼±  ┼├¯ _ ▒─  ¯ ┘  ├ ./_TOSSIM└¯ ┤┌ ._¯
±++ -_ ▒─  -°PIC  _ └ ┤ ┌ / ─ _/¯≤├¯__ └.¯ _ └ ┤ ┌ / ─ _/_ └.¯ _ └ ┤ ┌ / ─ _/├
¯__ └.¯ _ └ ┤ ┌ / ─ _/ -_┤──¯─├.¯ -┌_├ ++  -¯ _TOSSIM└¯ ┤┌ ._¯
  ¯─≤ ┼± P≤├ ¯┼ _ ─ ┬├  ┼├ ─°▒  TOSSIM.¯≤ °_¯Ⅰ ┌ /├¯__ └ ├¯ ┌¯ ▒┌  ─ ├¯─≤

*** S┤ __°┤┌┌≤ ┤ ┌├ ─ _ TOSSIM ┌ ─▒─≤.
|┤ ┤┼├¯_@|┤ ┤┼├¯_-├ ┼≤¯_:·/D _┐├¯─/R▒ ¯C¯┤┼├T¯L __°¯─ 5.2$ ¯≤├ ¯┼ ┤┼S └.¯≤
  1   2   -54.0
  2   1   -55.0
  1   3   -60.0
  3   1   -60.0
  2   3   -64.0
  3   2   -64.0
C─ ▒├ ┼± ┼¯ _  └¯ ┌ °¯─  1
C─ ▒├ ┼± ┼¯ _  └¯ ┌ °¯─  2
C─ ▒├ ┼± ┼¯ _  └¯ ┌ °¯─  3
DEBUG (1): A──┌ ▒├ ¯┼  ¯¯├ .
DEBUG (1): A──┌ ▒├ ¯┼  ¯¯├ .
DEBUG (2): A──┌ ▒├ ¯┼  ¯¯├ .
DEBUG (2): A──┌ ▒├ ¯┼  ¯¯├ .
DEBUG (3): A──┌ ▒├ ¯┼  ¯¯├ .
DEBUG (3): A──┌ ▒├ ¯┼  ¯¯├ .
DEBUG (1): *****************************************************
DEBUG (1): S ┼ ─ ID: 1
DEBUG (1): R▒ ¯C¯┤┼├T¯L _C: ¯▒┐ ├ _ ┼├.
DEBUG (1): T └ : 0:0:0.244150635
DEBUG (1): C ─ ─ T |├: KAMI_SAME
```

Except CTR and ECB, there are Cipher Block Chaining (CBC), Propagating Cipher Block Chaining (PCBC), Cipher Feedback (CFB) and Output Feedback (OFB) mode. I think OFB it more fitful for WSN, because by definition of self-synchronising cipher, if part of the ciphertext is lost (e.g. due to transmission errors), then the receiver will lose only some part of the original message (garbled content), and should be able to continue to correctly decrypt the rest of the blocks after processing some amount of input data [1].

Question 5. 2

Simple checksum is not enough because many combination have the same check sum.

MD5 is not enough because there are collisions, and it is a hash algorithm, hackers can intercept the MD5 message then index that message in a rainbow table then find out the original information.

CBCMAC is good because the message is encrypted with some block cipher algorithm in CBC mode to create a chain of blocks such that each block depends on the proper encryption of the previous block. This interdependence ensures that a change to any of the plaintext bits will cause the final encrypted block to change in a way that cannot be predicted or counteracted without knowing the key to the block cipher [2]. However there are still many way to attack due to using the CBCMAC incorrectly, such as using the same key for encryption and authentication, allowing the initialization vector to vary in value, and using predictable initialization vector [2].

The difference is HMAC uses hash functions for encrypting and CMAC block ciphers. For which is better in WSN, it depends on which device we use, how many resources we have on the devices, the library we have, and the requirements of security. If the requirement of security is high, we can use HMAC. It is really depend on the situation.

It is better not to use the same key.

There are ways to combine encryption and integrity protection together, it is called Authenticated Encryption (AE), there are different way to approach it, which are Encrypt-then-MAC (EtM) and Encrypt-and-MAC (E&M). EtM has highest security level in AE, the plaintext is encrypted first, then a MAC is produced based on the resulting ciphertext, The ciphertext and its MAC are sent together [3]. In E&M, a MAC is produced based on the plaintext, and the plaintext is encrypted without the MAC. The plaintext's MAC and the ciphertext are sent together.

There are different modes such as OCB and GCM.

OBC integrate a MAC into the operation of a block cipher, so it avoid a MAC for authentication and encryption for privacy. This results in lower computational cost compared to using separate encryption and authentication functions [4] .

Both security and performance of GCM is good, in performance GCM requires one block cipher operation and one 128-bit multiplication in the Galois field per each block (128 bit) of encrypted and authenticated data, so the block cipher operations are easily pipelined or parallelized. In security It is secure when it is used with a block cipher that is indistinguishable from a random permutation [5].

Lessons Learned & Conclusion

This lesson I learned a lot about encryption, and learned the simple use of it, in which I encounter many problems, such as wiring(because at first I cannot understand the relationship between these files), parameter problem(by trying many times and read the code annotation many times I eventually understand how to use these functions).

Do not forget every time the nesC code changed we need to recompile the code by "make iris sim"

And still, there are many problem remained as I mentioned in assignment 5.1 and 5.2. And in the end of the lab I cannot figure out why the receivers cannot decrypt the message properly. And I cannot figure out because there are some variables I cannot print it on the screen(when I try to access these variables, the code can go through the complier, but when it is running, there is always a segmentation fault, then the program stopped), so it is hard to debug which part is wrong, what is it inside a variable.

I learned many new concepts, such as block cipher, MAC. For block cipher there are many modes of operation, such as ECB, CBC, PCBC, etc. I learn the principle of these modes of operation and know the strength and short of each mode. Before this lab, I only know hash function and MD5.

MAC can be constructed from cryptographic hash functions(HMAC) or from block cipher algorithms(CMAC).

By learning these algorithm, I understood how the intruders attack the network, and how to prevent the attack. I learned encryption can protect our plaintext, but cannot prevent tampering. MAC can protect the integrity of the message from tampering.

And I learned there is a method AE can achieve encryption and integrity protection. There are 3 methods to realize it, they are literally Encrypt-then-MAC, Encrypt-and-MAC, and MAC-then-Encrypt. And there 6 different authenticated encryption modes, they have their own strength and short.

For the question of HMAC and CMAC which is better for WSN, at first I think HMAC is better because I saw many strength on it, such as security level and speed, but there different voice from the Internet, so I cannot say which is better.

References

[1]https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[2] https://en.wikipedia.org/wiki/CBC-MAC

[3] https://en.wikipedia.org/wiki/Authenticated_encryption

[4] https://en.wikipedia.org/wiki/OCB_mode

[5] https://en.wikipedia.org/wiki/Galois/Counter_Mode