Lab 3
Add an Encryption Layer to the IoT System

The objective of this lab is to learn an efficient way to encrypt the data transferred through the communication channel we created at the previous labs.

A core principle in the development of digital systems is that **security cannot be assumed** for a given system. The level of security is based on the complexity of the measures applied. For that reason, security in embedded systems is forshaken because the nature of end nodes dictates that they should remain low-cost and power and area efficient. As a result, the vast number of devices an IoT network can consist of creates an extensive attack surface, putting personal data at risk. Consider the case of a health monitoring system, which handles highly sensitive data.

An extensively examined algorithm for IoT use is the Advanced Encryption Standard (AES). In the following sections we will examine the core principles of AES and develop it from scratch to be used in our embedded system.

AES Introduction

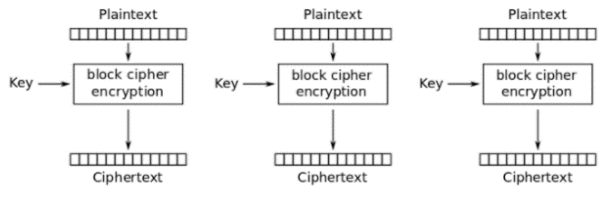
The algorithm opted to be used in the AES standard is called **Rijndael Algorithm**, which is a symmetric block cipher that can offer encryption and decryption operationality.

- Symmetric Cipher: Requires a single value (key) that should remain secret for the success of the intended operation
- Block Cipher: The data are encrypted in blocks of specific length. Modes of operation can be defined in order to specify the interaction of those blocks.

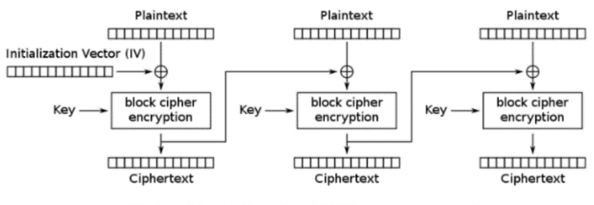
In AES, the secret key and the given data (plaintext) need to be of specific length (128-bit, 192-bit or 256-bit AES), resulting in an output (ciphertext) of the same length. **AES-128** will be used.

The simplest mode of use is **Electronic Codebook (ECB) mode**, meaning that the data is separated into blocks of fixed length (in the case of AES-128, blocks are of 16 bytes length) that will be encrypted handled independently of one another.

What are the advantages and disadvantages of that? Consider another mode, CBC.



Electronic Codebook (ECB) mode encryption



Cipher Block Chaining (CBC) mode encryption

Is AES-128 ECB secure enough?

From software perspective, yes. As mentioned before, the success of the algorithm depends on the secrecy of the key. In order to “break” the cipher, you need to obtain the correct key. Assume the simplest attack, brute force attack. The worst case requires the examination of the total number of keys. If a key consists of 128-bits, a total of 2^{128} possible keys need to be examined.

Better formulated attacks can in theory lower the complexity to 2^{126}. Quantum attacks will probably break the AES-128 (meaning AES-192 and AES-256 are no longer safe), but they do not yet pose a threat. For now, AES-128 ECB **achieves computational security**, meaning the security is based on the fact that current technology is not capable enough to break the algorithm at an acceptable time.

When applied to hardware, the algorithm can be easily broken with an attack aiming the physical vulnerabilities of the device, prompting to the development of specific countermeasures.

The Rijndael Algorithm

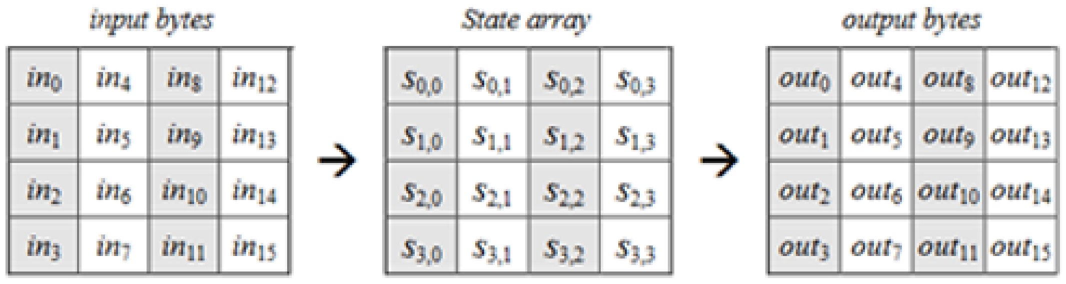
The Rijndael algorithm[1] is a very good application of **confusion and diffusion** principle. Confusion is achieved when the relationship between the secret key and the ciphertext is not distinguishable. Diffusion means that a modification over a single bit of the plaintext can statistically affect half the bits of the ciphertext. Both are simply realized though permutation and substitution operations (substitution-permutation network).

The algorithm defines three important values:

- N_r : Number of rounds the operations will perform over the data.
- N_k : The key-length in bits, divided by 32.
- N_b : The length of the word.

In the case of AES-128, N_r=10 and N_k= 4. N_b depends on the system’s architecture and is most commonly equal to 4.

The data of a block, separated in bytes, are handled as a 2-dimensional array (state) of N_b rows and N_k columns, placed column-wise.



[1] https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf

Before the encryption steps take place, a **Key Expansion** operation is performed. The secret key is used to generate a Nr+1 number of keys (round keys), with a total of (Nr+1)*Block_Size bytes.

1. IoT Node – Project Setup

Firstly, create a new project, under the name Lab_3. You can reuse the setup you created on the previous lab exercise.

Copy the aes.cpp and aes.h files contained in your handout to your current project, in the same location as main.cpp. Include aes.h library in main.cpp with the appropriate command.

In previous labs, the required data were sent and received as a string. The algorithm you are going to develop uses uint8 arrays of 16 bytes for the plaintext, the key and the ciphertext.

Go to main.cpp and:

1. Create the required data structures – key, plaintext and ciphertext - in main.cpp.
   The key is given.
2. Add the data of your choice in your plaintext structure as uint8 type
3. Encrypt those data. *You can verify the correct passing of plaintext to Matlab before applying the encryption*
4. Send to serial independently of what the node reads – data read will only operate as trigger.

2. IoT Node - Develop AES from scratch

The encryption mechanism **AES_ECB_encrypt()** uses the plaintext, the key and the block size in bytes to write the proper data on the ciphertext. The plaintext (input) is copied to the ciphertext (output), it is parsed as state (16 bytes array into 4x4 bytes matrix) and the operations are performed over it.

You will code the functionality for AddRoundKey and SubBytes only.

A description of the desired functionality is given below:

- **AddRoundKey(uint8_t round)**: A Round Key (16 bytes of the 176 array) is applied to the State (4x4 bytes matrix), with a simple XOR operation.

  Example for round = 0:
  - *(*state) [0][0] ^= RoundKey[0]*
  - *(*state) [0][1] ^= RoundKey[1]*
  - *(*state) [1][0] ^= RoundKey[4]*

  Example for the round = 1:
  - *(*state) [0][0] ^= RoundKey[16]*
  - *(*state) [0][1] ^= RoundKey[17]*
  - *(*state) [1][0] ^= RoundKey[20]*

  The first AddRoundKey is performed outside the round operations.

- **SubBytes()**: A byte substitution follows, applied over each of the State's byte separately. Most commonly, a substitution (look-up) table is used, of size 16x16 (=256 bytes). In this implementation, the input byte serves as the index the output value is located.

  Example, SubBytes(0) = 0x63 = 99, SubBytes(1) = 0x7C = 124, SubBytes(17) = 0xCA = 202, given the following sbox:

```
static const uint8_t sbox[256] = {
0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};
```

- **ShiftRows()** : It is applied over the <u>rows</u> of the state. It shifts their content cyclically, on an offset determined by Nb. the first row is not shifted (shifted by 0). The last row is shifted by Nb-1. Apply the pattern for all state's rows.

- **MixColumns()**: It is applied over the <u>columns</u> of the state this time. Considering the column as a matrix, each of is multiplied by a fixed matrix, producing a new matrix. **The last round skips the MixColumns step.**

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Hint 1: 3=2+1 => $3a_i = 2a_i + a_i$
Hint 2: There is an easier way to perform multiplication with 2. Code it in xtime() function.
Hint 3: Addition is performed modulo 2. This corresponds to known Boolean operation.

3. IoT Node - Encrypt data

Having the code above, encrypt the plaintext data and send the ciphertext to the hub through the serial.

4. Matlab - Decrypt data

Using the setup from the previous lab:

1. Receive the encrypted data. Take into account that you need to read bytes
2. Apply AES decryption (The code is given). Pay attention to the inputs' format. How do we know the data are decrypted correctly?

Αναφορά Εργαστηρίου 3

Για τη βαθμολόγησή σας είναι υποχρεωτική η παράδοση αναφοράς, η οποία θα έχει αυστηρή προθεσμία το τέλος της ημέρας του επόμενου εργαστηρίου .

Η αναφορά θα πρέπει να περιέχει αναλυτικά όλα τα βήματα που ακολουθήσατε στο εργαστήριο και τη δικαιολόγησή τους.

Επιπλέον ερωτήσεις για την αναφορά του εργαστηρίου 3

1. **Ποιό είναι το χρονικό overhead σε σχέση με την προηγούμενη υλοποίηση;**
2. **Ο αλγόριθμος κρυπτογράφησης που υλοποιήσατε ονομάζεται tinyAES. Συγκρίνετε με τον AES και διατυπώστε τις διαφορές.**
3. **Βάσει του παραπάνω, σε πιο τομέα ωφελεί η χρήση του tinyAES τα ενσωματωμένα συστήματα;**
4. **Εμβαθύνετε στο SubBytes. Εξηγήστε περιγραφικά τους υπολογισμούς πίσω από την διαμόρφωση των look-up tables. Πότε προτιμάται η χρήση πινάκων αντικατάστασης και πότε των υπολογισμών;**

Further Reading

1. Serra LFD, Goncalves PGB, Lopes Frazao LA, Antunes MJG. Performance analysis of AES encryption operation modes for IoT devices. In: *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE; 2021:1-6. doi:10.23919/CISTI52073.2021.9476528

2. Munoz PS, Tran N, Craig B, Dezfouli B, Liu Y. Analyzing the Resource Utilization of AES Encryption on IoT Devices. In: *2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE; 2018:1200-1207. doi:10.23919/APSIPA.2018.8659779