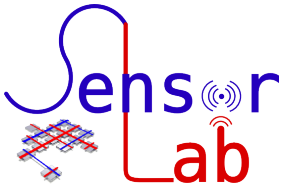
Faculty of Mathematics and Computer Science

– Georg-August University Göttingen –

Telematics Group

Lab 5 - Encrypted Communication (Symmetric)

Sensorlab



Contents

1 Introduction

Confidential communication in wireless sensor networks can be important. For this reason, this lab will explain the basic use of symmetric cryptographic techniques in WSNs, which will allow encrypting messages sent by sensor nodes. A short look at providing integrity protection will also be taken.
Specifically, this lab will mostly be centered around block ciphers and modes of block cipher operation. For information on the operation of block ciphers and cipher modes, please read:
`http://en.wikipedia.org/wiki/Block_cipher`
`http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation`

2 Setup

First, you will need to get the code for some cryptographic components for TinyOS.
`http://user.informatik.uni-goettingen.de/~sensorlab/code/crypto.zip`
You can download and set it up like this:

```
cd YourApplication/
wget http://user.informatik.uni-goettingen.de/~sensorlab/code/crypto.zip
unzip crypto.zip
```

3 Interfaces

Three interfaces are supplied by the cryptographic components:

```
/* For detailled parameter information, check the corresponding
   .nc files. */
interface BlockCipher
{
  command error_t init(CipherContext * context,
                       uint8_t blockSize, uint8_t keySize, uint8_t * key);
  command error_t encrypt(CipherContext * context,
                          uint8_t * plainBlock, uint8_t * cipherBlock);
  command error_t decrypt(CipherContext * context,
                          uint8_t * cipherBlock, uint8_t * plainBlock);
}
interface BlockCipherInfo
{
  command uint8_t getPreferredBlockSize();
  command uint8_t getMaxKeyLength();
  command bool getCanDecrypt();
}
interface BlockCipherMode
{
  command error_t init(CipherModeContext * context,
                       uint8_t keySize, uint8_t * key);
```

```
  command error_t encrypt(CipherModeContext * context,
                          uint8_t * plainText, uint8_t * cipherText,
                          uint16_t numBytes, uint8_t * IV);
  command error_t decrypt(CipherModeContext * context,
                          uint8_t * cipherBlock, uint8_t * plainBlock,
                          uint16_t numBytes, uint8_t * IV);
  /* The following are not important for this lab: */
  command error_t initIncrementalDecrypt (CipherModeContext * context,
                                          uint8_t * IV,
                                          uint16_t length);
  command error_t incrementalDecrypt (CipherModeContext * context,
                                      uint8_t * ciphertext,
                                      uint8_t * plaintext,
                                      uint16_t length,
                                      uint16_t * done);
}
```

Two implementations of different block ciphers are included, both of which provide the `BlockCipher` and `BlockCipherInfo` interfaces. The module `XTEAM` provides the XTEA cipher, while the `AES128M` module provides the AES-128 cipher. The `CTRModeM` module provides te `BlockCipherMode` interfaces and implements the Counter block cipher mode.
The `BlockCipherInfo` interface provides a program with information about the cipher module it was wired to. It allows requesting information such as the maximum permissible encryption key length in bytes, the preferred block size of the cipher in bytes and whether the cipher has a built in decryption function.
The `BlockCipher` interface provides raw access to a block cipher's block encryption and decryption functions. Before using these, a cipher context struct has to be initialized with the key. The cipher can then do any necessary key setup (or copy the key into the context struct to do so later), allowing the resulting cipher state to be used for subsequent calls to the encrypt and decrypt functions.
The `BlockCipherMode` interface provides access to a block cipher on a higher level. The `CTRModeM` requires a block cipher module to be wired to the `BlockCipher` and `BlockCipherInfo` interfaces it uses. After initializing the corresponding context with a key, it can be used to encrypt or decrypt plaintext or ciphertext buffers of a given length with a specified initialization vector.

3.1 `CTRModeM` implementation notes

For the `CTRModeM` module, certain behavior regarding the initialization buffer should be noted.
In the case of encryption, if the IV pointer is NULL, the module assumes it should use its internal counter value. If an IV is given the first time the

encrypt function is used with a context, it will set its internal counter to this value. At any later pointer, instead the specified IV buffer is overwritten by the relevant internal counter state. After encryption, the internal counter state is updated with the newest value.
For decryption, again, the internal counter state is used if the provided IV buffer is NULL. Otherwise, the value given in the IV buffer is used. After decryption, correspondingly, either the internal counter state, or the provided buffer is updated with the new counter state.
The module also provides the commands `get_counter` and `set_counter`. However, since these should only be used rarely, they are provided outside of any regular interfaces and have to be wired up manually.
For details regarding the behavior described above, please consult the source code.

4 Assignments

4.1 Question 5.1: Ciphers (5 Points)

Looking at the implementations of the `BlockCipherInfo` interface of both ciphers. What differences do you find? Do you think a block cipher could still be useful without a decryption function? If so, why? Otherwise, why not?

4.2 Assignment 5.1: Simple encryption/decryption (25 Points)

Build an application that uses the `BlockCipher` interface directly, to encrypt and decrypt data. Assuming that the amount of data does not exactly fit the block size of the cipher, how can you deal with it? Assuming you just concatenate encrypted blocks, what is this called, and what problems do exist with this approach? How can these problems be avoided?
Regarding your implementation, you can encrypt and decrypt whatever string you like. Radio communication is not necessary. It may be a good idea to use TOSSIM for simulation in this lab, so you can easily output things using `dbg()` (e.g. output like "Decrypted data matches original data: Success").

Hint: Take a look at the functions `memcpy()`, `memset()` and `memcmp()`, they might help you handling your data arrays.

4.3 Assignment 5.2: Using CTR mode (15 Points)

Modify your application to use the `BlockCipherMode` interface instead of using `BlockCipher` directly, by using the `CTRModeM` module. Please note that you will not only have to wire the `CTRModeM` module to the `BlockCipherMode` module, but also one of the cipher modules to the `CTRModeM` module.
Does there exist a problem when using Counter mode in an environment where packets may get lost during transmission and if so what is it? Can initialization vectors/counter values be safely reused with Counter mode? If so, why? If not, why not? Does Counter mode impose any practical restrictions on message length?
Name at least two other modes of block cipher operation except CTR and ECB. How do they differ from Counter mode? Do you think their properties make them more, less or equally suitable for use on wireless sensor motes?

4.4 Question 5.2: Integrity protection (5 Points)

So far, we have only looked at encryption and decryption, which provides confidentiality to our data. However, an attacker might also attempt to tamper with our encrypted data. Investigate some methods to protect the data, such as different MAC methods.
Why is a simple checksum or MD5 sum insufficient?
Is CBCMAC a good way to achieve integrity protection?
Block ciphers are often fast and readily available. When comparing HMAC and CMAC, which do you suspect will work better in a WSN scenario? What are the differences between them?
Is it a good idea to use the same key for both encryption and the MAC function?
Are there ways to combine encryption and integrity protection? Name at least two examples. Is a single key sufficient in these cases? Do different approaches differ with respect to their performance characteristics? Will such an approach be usually faster, slower or equal to doing encryption and MAC calculation as separate steps?