Application Note:
SimpliciTI Security

Document Number: SWRA278

Author: Larry Friedman

Texas Instruments, Inc.
San Diego, California USA

Version	Description	Date
1.00	General release	03/01/2009
1.10	Updated title page, modified formatting	03/24/2009

Table of Contents

1. Introduction 1
2. References 1
3. Summary 1
4. Details 2
4.1. Message Authentication Code (MAC) 2
4.2. Payload encryption and decryption 2
4.3. Security parameters 3
4.3.1. Initialization vector (IV) 3
4.3.2. Key 3
4.3.3. Message Authentication constant 4
4.3.4. Counters 4
4.3.5. Message Authentication Code (MAC) 5
4.4. Frame format 5
4.5. Endianess and alignment 6

List of Figures

Figure 1: SimpliciTI encryption schematic 2
Figure 2: SimpliciTI decryption schematic 3
Figure 3: SimpliciTI frame format with Security enabled. 6
Figure 4: Network header security bytes detail 6

1. Introduction

SimpliciTI Release 1.1.0 will introduce a security capability. The solution will be a software solution applicable across all supported platforms.

The application payload is encrypted. With the exception of two new encryption infrastructure bytes added to the network header, the remainder of the SimpliciTI frame will remain in plain text.

Enabling security will modify the SimpliciTI frame as there is additional overhead required to support the encryption. Three bytes are added. These bytes are logically part of the network header. One of them is plain text and the other two are encrypted.

The encryption algorithm is based on a scheme available in the public domain. The encryption mode is a modified CTR (counter) mode as described in the sections below.

Details are provided in the following sections.

2. References

[1] *SimpliciTI Specification,* Texas Instruments, Inc.

[2] *Tea extensions,* Roger M. Needham and David J. Wheeler, Technical report, Computer Laboratory, University of Cambridge, October 1997 (PDF available on Internet)

3. Summary

Encryption is enabled as a build-time option for each device. If the macro **`SMPL_SECURE`** is defined in the **`smpl_nwk_config.dat`** file security is enabled. The infrastructure in support of security is contained entirely in the network layer. There is no impact on the application layer so the application APIs do not change.

Each call from an application to send a frame will result in the application payload being encrypted before the frame is sent. On receipt the payload is decrypted before being made available to the application. The same mechanism is used for both user applications and network applications.

There are 3 components of the encryption scheme that can be used to maintain security: the 128 bit key, a 32-bit Initialization Vector (IV) and a 32-bit counter. The key and the IV are fixed at build time. The counter starting value is exchanged during the link session between the user application peers. Independent counter values are maintained for sending and receiving on the connection. The counter value for the network applications are handled differently because they are not connection-based. This is discussed in more detail below.

A modified CTR mode is used here. The scheme encrypts 64-bit blocks using an encryption scheme available in the public domain (XTEA: E**X**tended **T**iny **E**ncryption **A**lgorithm). The 64-bit block is a concatenation of the 32-bit IV and a 32-bit counter. Changing the counter for each encryption guarantees a unique block for each encryption action. The key length used in the encryption is 128 bits.

An exclusive bitwise logical OR is done between the resulting 64-bit cipher block and the (next) 64 bits of plain text to be sent. If the remaining plain text is less than 64 bits the extra cipher block is discarded. If the plain text remains the next cipher block is created incrementing the counter.

4. Details

4.1. Message Authentication Code (MAC)

The SimpliciTI security method includes a minimal Message Authentication object. This two byte object consists of a byte of fixed value and a byte that represents a frame check sequence (FCS) calculated over the application payload. The default FCS is a simple logical sum with a starting value of 0. The fixed value byte is included in the FCS calculation.

This two-byte object is pre-pended to the application payload and is encrypted as part of the payload as described below.

4.2. Payload encryption and decryption

The following Figure describes the send side action.

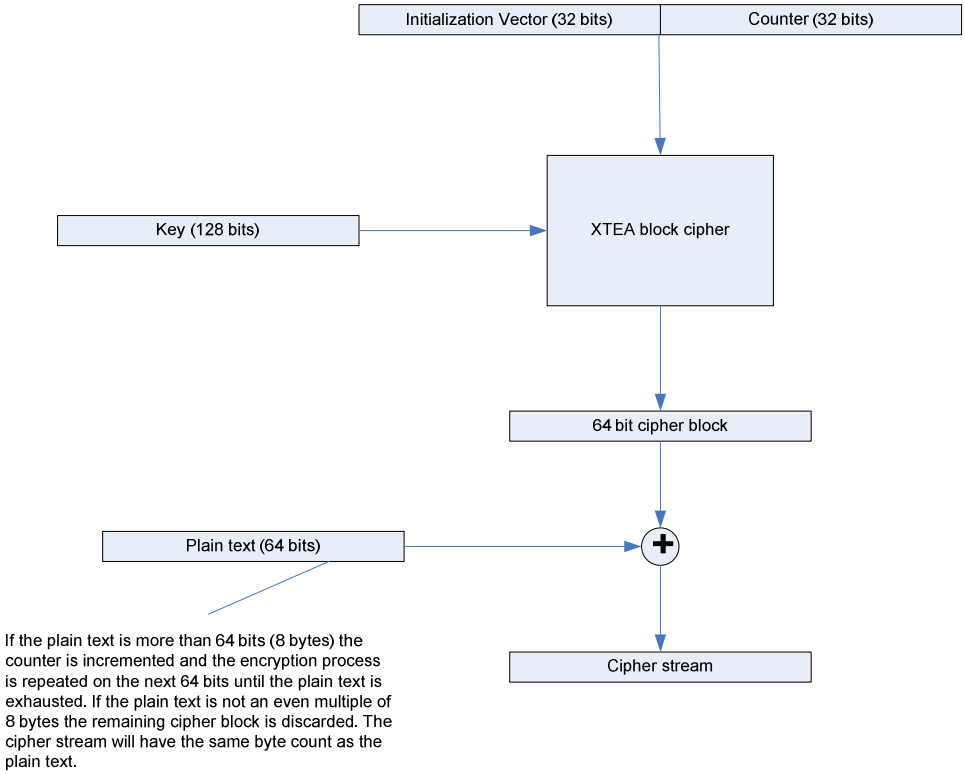

Figure 1: SimpliciTI encryption schematic

Conceptually it is the 64-bit concatenation of the IV and the counter that is being encrypted. It is guaranteed to change each block because the counter changes each block. The counter does not repeat for a very long time so the cipher block will not repeat. The plain text is simply exclusively or'ed with the cipher block to produce the encrypted payload.

Decryption with this method looks similar:

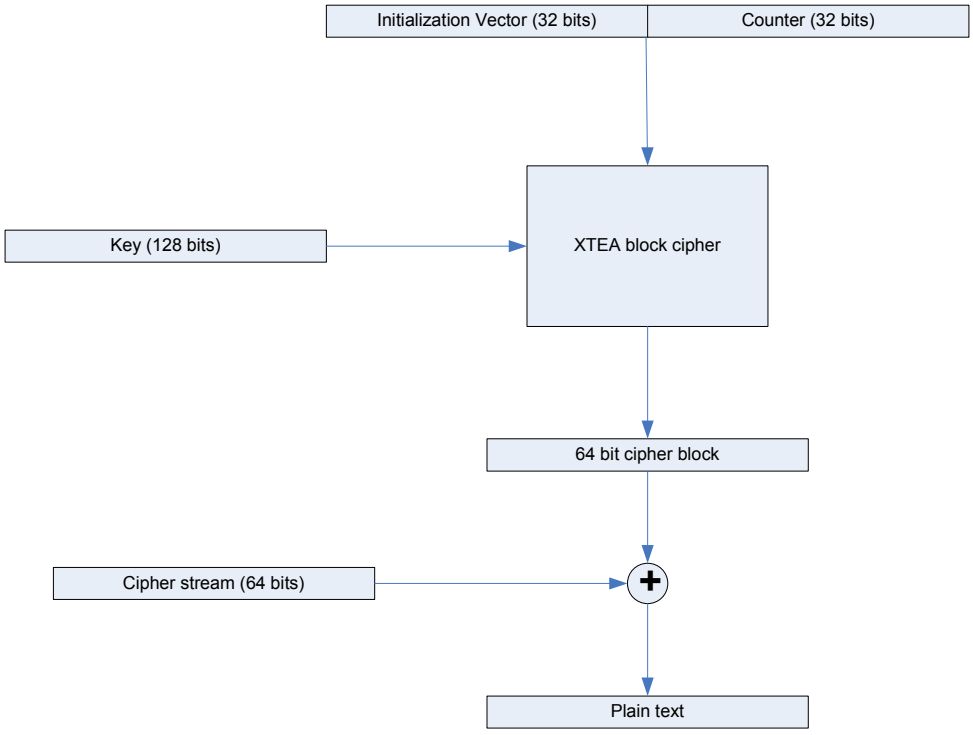

Figure 2: SimpliciTI decryption schematic

Decryption is accomplished by encrypting the same block as the encryption side. The resulting cipher block is then exclusively or'ed with the received cipher stream to reproduce the original plain text. Since the cipher stream matches the length of the original plain text the same procedures are used to rectify streams that are not 64 bit multiples.

Clearly this scheme depends on keeping the counters synchronized. This will be discussed in a following section.

The security parameters are discussed below.

4.3. Security parameters

4.3.1. Initialization vector (IV)

This is a 32-bit unsigned object set at build time. It is declared and defined in the file **`nwk_security.c`**. User can change this as required.

4.3.2. Key

This 128 bit value is set at build time. It should be initialized as a 16 byte array of characters. The encryption algorithm works with 32 bit unsigned long objects. The security initialization code assumes that the key is a character array and does byte swapping as necessary to accommodate native endian

attribute of the host microcontroller. This must be done to guarantee that microcontrollers with different native endian representations give the same encryption results.[1]

The key initialization also occurs in the file **`nwk_security.c`**.

4.3.3. Message Authentication constant

As described in Section 4.1 the MAC contains a constant part. This one byte object is declared and defined in the file **`nwk_security.c`**. The FCS scheme is also defined in this file.

4.3.4. Counters

The counter is a 32 bit unsigned long object. For connection based applications initial values for the counters for each connection are generated at run time. These connections essentially represent User applications. The initial values are exchanged in the link session when the connection is established.[2]

For network applications the counter values are derived differently since the network applications are not connection based.

4.3.4.1. User applications

The initial 4 byte values for User application counters are negotiated during the Link session. The connection established is bi-directional and a separate counter is used for each direction. This keeps things cleaner if each direction is used by each peer asynchronously. Independent channels require independent counters.

Because SimpliciTI does not support guaranteed delivery there must be a mechanism to synchronize the counters. Otherwise correct decryption could not occur. The synchronization is done by supplying a counter hint in the frame containing the encrypted payload. The hint is a single byte representing the least significant byte (lsb) of the 4 byte counter value used to encrypt the accompanying payload. It is sent in the clear over the air.

The count hint should have the same value as the lsb of the local copy of the counter for that connection. If it does the entire local counter value is then used to decrypt the payload. Note that even if the hint and local lsb bytes match the decryption may still be invalid. This can happen if the MAC fails its check. The MAC guards against replay attacks.

If the bytes do not match an attempt is made to reconcile the counts. The received byte could differ for a number of reasons:

- The frame could be a valid late arrival (e.g., a replay from a Range Extender)
- The local device could have missed frames but the current one is valid
- The frame could be a rogue, for example, from a replay attack

In the first two cases the frame is valid but the decryption will be guaranteed to fail because the counters do not match. The local device must then use the hint as the lsb and continue. In the last case it would be a mistake to permit the counter hint to govern. The problem is to distinguish among these alternatives.

---

[1] Initializing the key as 4 unsigned longs would remove the byte swapping requirement. However, initializing the key by using a character array is more convenient and a common method used with long keys.

[2] See Section 4.3.4.2 for an exception when the Unconnected User Datagram Link ID is used.

The current design places a window around the expected counter hint (the lsb of the local counter copy). If the counter hint is not within this window the frame is rejected as a rogue.

If the received frame counter hint falls within this window the counter hint is used to decide whether the frame is late or frames were missed. If the frame is late the frame is ignored. If frames were missed the counter hint is used as the lsb of the 4 byte local counter and the decryption sequence proceeds. Note that the frame may still be deemed invalid because of failure of the MAC.

In making a decision about the acceptability of the frame the processing of the counter hint must account for the counter wrap around 0.

If a frame is decrypted and accepted at the network layer the counter associated with that connection is updated based on the counter value that obtains after the decryption. In this way the lsb of the local counter now reflects the expected value of the counter hint in the next frame received for that connection.

It is possible to incorrectly reject a frame or to incorrectly accept one. These cases are unlikely. Application payload content can be embellished to add sanity checks at the application layer as additional checks.

The size of the window is defined by the macro **`CTR_WINDOW`** defined in the file **`nwk_security.c`**. The default value in the SimpliciTI distribution is 255.

4.3.4.2. Network applications

Network applications and the Unconnected User Datagram Link ID (UUD) are not connection based. It is not possible to maintain a coherent 4 byte counter.

Instead the frames sent to a network application and the UUD populate the counter hint with a random value. This value is used by the recipient to decrypt the payload. The hint is used as the lsb of a counter whose remaining 3 bytes are 0. The validity of the frame and defense against rogue frames comes from the protection of the values of the IV and the key. In addition there is the protection afforded by the MAC.

This is how there can be some security around the exchange of counter starting values when the Link session is engaged. The Link session takes place on the Link Network application port.

4.3.5. Message Authentication Code (MAC)

To help defend against rogue, replayed frames two additional objects are added to the frame. A one-byte frame check sequence (FCS) and a one byte fixed message authentication value are each placed in front of the application payload. They are both encrypted during the securing of the frame. Together these two bytes are referred to as the Message Authentication Code (MAC).

When the payload is decrypted the value of each of the MAC objects is compared with the expected value. If they do not match the frame is ignored. In this case the current value for the local counter is not changed.

The fixed byte value is declared and defined in the file **`nwk_security.c`**.

4.4. Frame format

To support encryption two changes are made to the frame format. First, when frames are encrypted the encryption valid bit is set. See Section 6.5.4 in **Error! Reference source not found.**.

Secondly, three bytes are added to the frame. These three bytes are the counter hint and the 2 byte MAC. The MAC encompasses both a fixed value byte and a FCS. See Section 4.3.5 above. Note that this decreases the valid maximum application payload allowable by 3.

The frame format for encrypted frames is shown below:

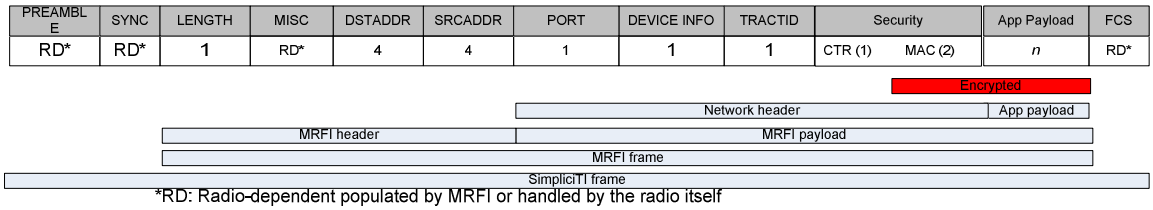

Figure 3: SimpliciTI frame format with Security enabled.

Details of the 3 bytes supporting security and their relationship to the application payload are shown below. The fixed value MAC byte is placed adjacent to the application payload and is included in the FCS calculation.

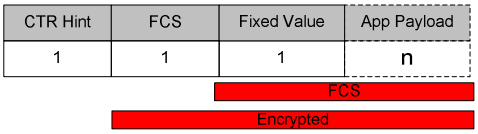

Figure 4: Network header security bytes detail

4.5. Endianess and alignment

Multi-byte quantities are being transported to support Security. All multiple byte quantities are converted to Network byte order (little endian) before being placed in frames. This is true of the counter values in the Link frames on behalf of the Link network application. User should be aware of this issue when setting the encryption key. See Section 4.3.2. This is of particular note if participating platforms support different endian representations.

In addition, object alignment is accounted for when creating and parsing messages in the network applications. This is not an issue for the 8 bit 8051 core targets. But it is an issue for the 16 bit MSP430 core targets.