Freescale Variable Key Security Protocol Transmitter User's Guide

by: Ioseph Martínez and Christian Michel
Applications Engineering - RTAC Americas

1 Introduction

The software explained in this document allows implementing a complete remote keyless entry (RKE) system. The variable key security protocol (VKSP) implements secure communication using one-way authentication with 128 bits of encryption. The VKSP software library is divided in two main blocks, transmitter and receiver. This document focuses only on the transmitter part.

2 VKSP Overview

VKSP is a protocol intended mainly for RKE systems. It sends commands rather than information. The commands sent through the communications link are visible for any device monitoring the communications channel. VKSP carries two main tasks: the generation of authentic messages and the verification of received messages. These tasks are abstracted from the application layer.

Contents

© Freescale Semiconductor, Inc., 2008. All rights reserved.

freescale semiconductor

A transmission frame is generated with every command sent. The transmission frame has two main parts: the data section and the message authentication code (MAC) section.

The data section is sent un-encrypted to the receiver and has the following information:

- Transmitter ID: This is a three-byte identifier of the transmitter device.
- Command: This is the one-byte command that indicates the to receiver what action to perform.
- Variable Key: This is a four-byte value incremented with time and ensures that the messages received were not previously sent.
- Message authentication code: This 8-byte code authenticates messages on the receiver side.

The message authentication code ensures that the data information is authentic. The MAC is generated automatically by the VKSP transmitter driver. To generate a MAC, the drivers use local keys generated internally and that cannot be accessed from external functions for security reasons. The next figure depicts how the transmission frame is generated.

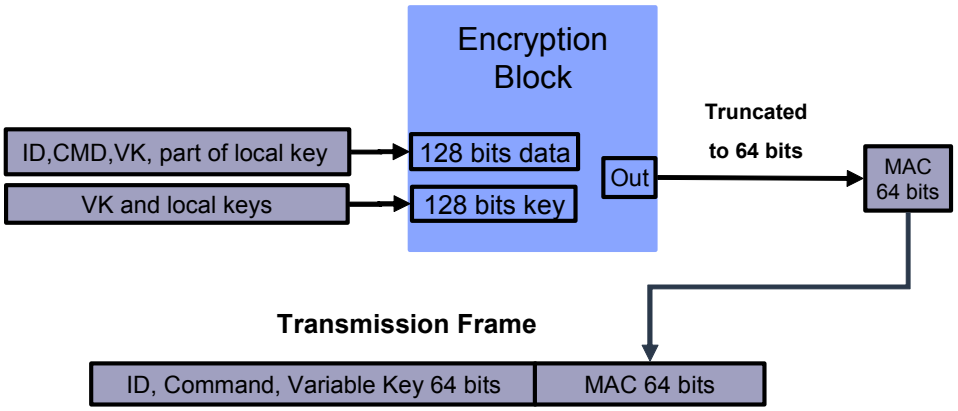

Figure 1. Transmission Frame Generation

The steps taken to verify the validity of a message are the following:

1. The ID data from the incoming message is extracted. This ID is looked-up in the receiver database. If the ID is found in the receiver database, a local key and a variable key associated with that ID are fetched.
2. The received variable key must be greater than the stored variable key. This step ensures that any re-transmitted frame (a frame that was already sent before) is not accepted.
3. The authentication is performed. A message authentication code (MAC) is generated from the received data, local data, and local keys. This generated MAC is compared to the received MAC from the received message. If these two MACs are equal, the command is accepted.

The authentication process is illustrated in the Figure 2.

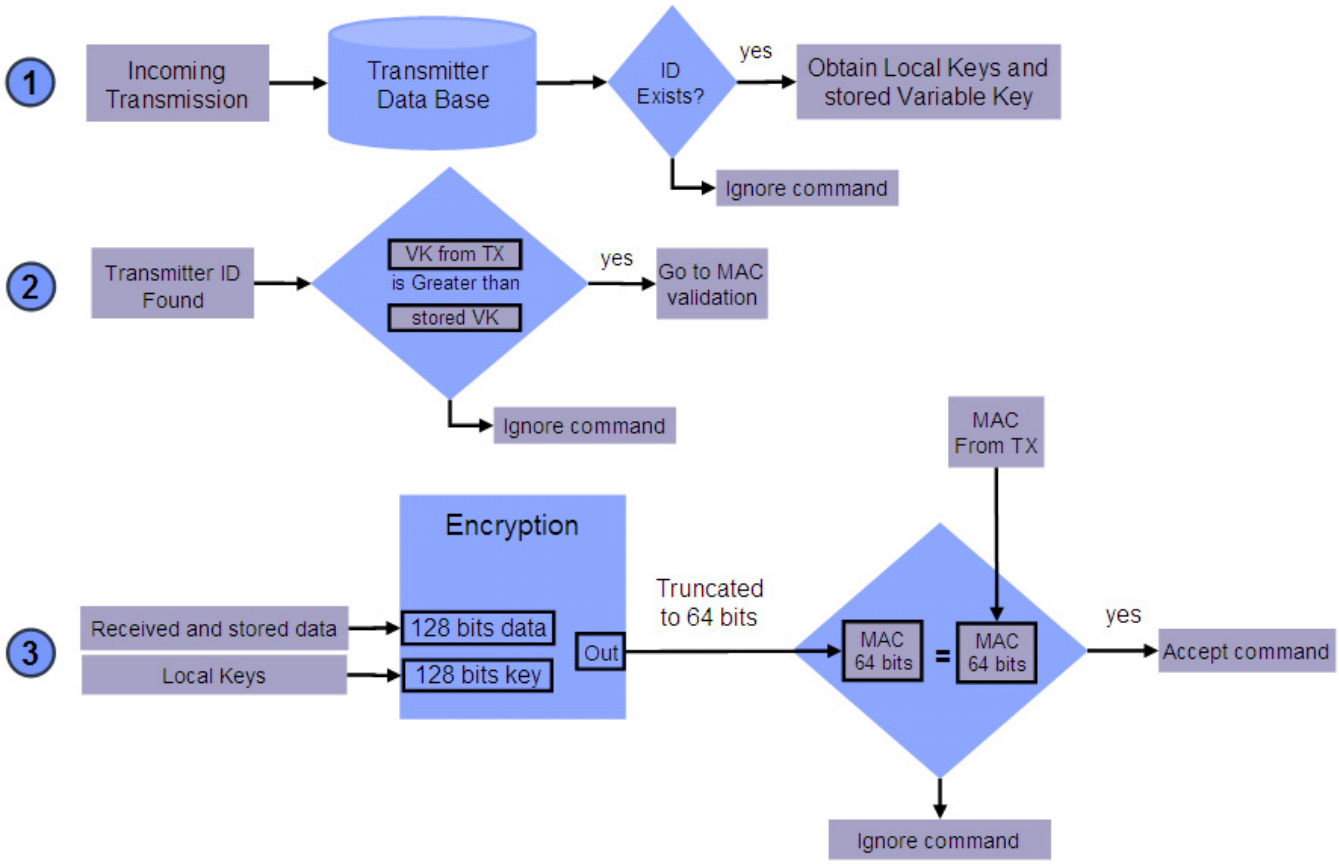

Figure 2. Normal Transmission Validation

2.1 Associating a Transmitter with a Receiver

A learning sequence is a process that must be performed each time a new transmitter is registered in the receiver system. For example, if a car user acquires a new transmitter key fob, it is necessary to register that key fob into the database of the car. This is done by performing a learning sequence. When the car receiver system receives the learning sequence, it stores the ID and local keys that pertain to that specific transmitter.

Below is how a learning sequence is performed. These steps are also explained in Figure 3.

- The VKSP transmitter has a source of data from a pseudo random number generator (PRNG), which generates the local keys.
- Inside the integrated encoder, this pseudo random data is scrambled and two frames are generated. This information is called agreement info. The frames are depicted in Figure 4.
  - Learning Frame 1: Contains a 3-byte header with ID information and a 1-byte field identifying the frame as a learning frame. Value 0xFE in this field indicates this is the learning frame number 1. The rest of the frame has agreement information used in the receiver and transmitter to generate a common local key for that specific ID. The length of the learning frame body is 12 bytes.

— Learning Frame 2: Contains a 3-byte header with ID information and a 1-byte field with value 0xFF identifying the frame as the learning frame number 2. The rest of the frame has agreement information (4 bytes) and an 8-byte MAC.

- First, the receiver checks if there is a secure environment activated. This means the receiver must be in a mode that allows learning processes. For security reasons, it is not possible to perform the learning process in the receiver at any time.
- The integrated decoder receives the two frames. If the OEM key (This is the manufacturer key that must be shared by all the authentic devices) is the same on the transmitter and receiver, the information is saved on a transmitter database for future use.

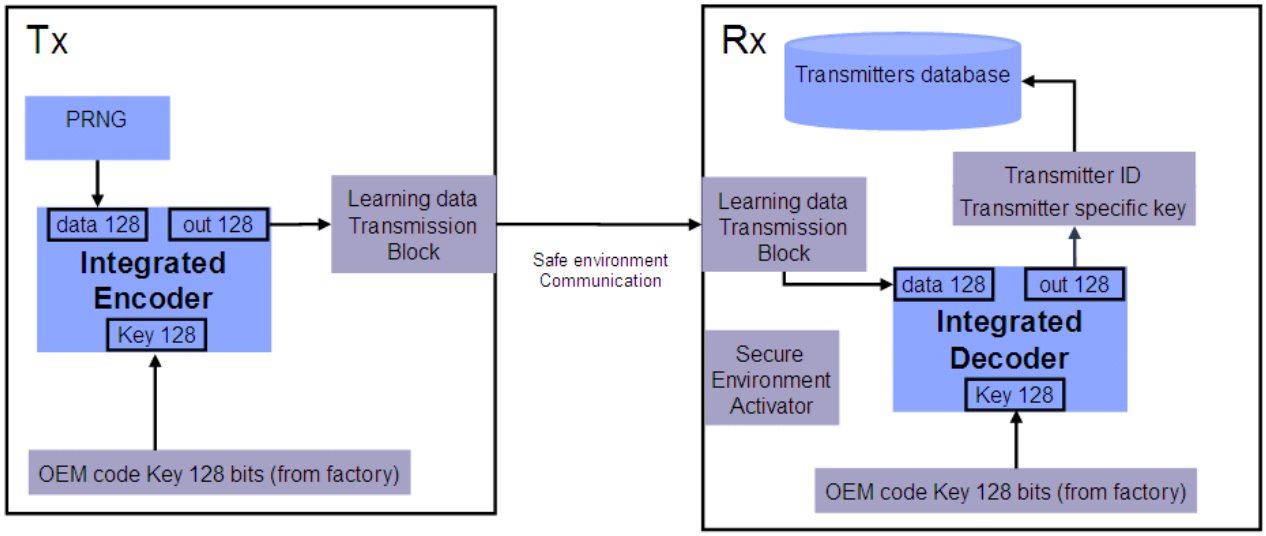

Figure 3. Learning Sequence

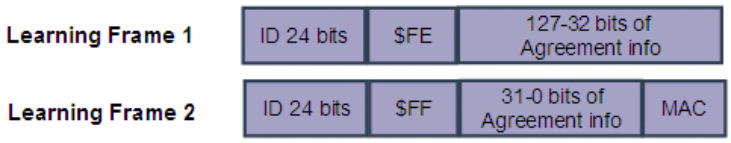

Figure 4. Learning Frames

3 VKSP Transmitter Driver Overview

The VKSP transmitter driver manages the generation of VKSP frames. The routines provided here initialize the VKSP module, generate the learning sequences, create the normal command frames, and manage the variable key (or counter) increments.

The driver is developed for the S08 family.

This library does not use any microcontroller hardware peripherals directly. Its flexibility allows the final application to use any drivers for encryption, random number generation, and non-volatile memory access.

The Figure 5 depicts the library interface and the communication with other modules.

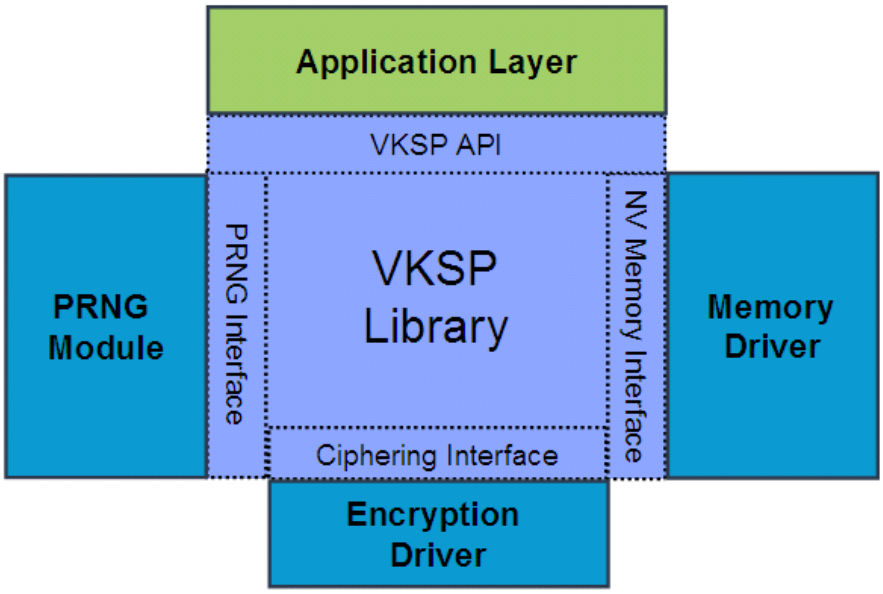

Figure 5. VKSP Transmitter Block Interaction Diagram

The main block in Figure 5 is the VKSP Library block. The VKSP block interacts with other layers through an interface. These interfaces offer flexibility and allow the user to select different customized drivers.

- PRNG Interface: Interface to obtain pseudo random numbers. Depending on the hardware used, these numbers can be generated by software or hardware.
- Ciphering Interface: Interface to encrypt data. Allows using any encryption algorithm.
- Non-Volatile Memory Interface: Interface used to store non-volatile data. Non volatile data may be stored in memories such as flash or EEPROM.

The details about the requirements of each driver are explained in the interfaces section.

The VKSP API allows the user application to initialize, generate messages, and perform a learning sequence.

4 Configuration of the VKSP Transmitter Driver

The vksp_tx_cfg.h file contains all the initialization parameters that must be adapted for a specific application. This is the only file that you should modify. These parameters are linker time effective. This means the VKSP core could already be compiled and provided as object code and changes on the parameters continue to have effect.

Table 1 shows all the user defined values for the VKSP transmitter driver:

Table 1. Configuration Parameters

MAX_LEARN_COUNT	Defines the maximum number of learning sequences that can be performed during the life of the application. The learning counter variable is a value stored in non-volatile memory and checked below the range of MAX_LEARN_COUNT. Example of usage: `#define MAX_LEARN_COUNT (64)`

VKSPTX_OEM_KEY	This is the 16 bytes manufacturer key that must be the same in all the devices intended to be used together. If a transmitter and a receiver have different OEM keys, it is not possible to execute a successful learning sequence.
	
	Example of usage:
	
	`#define VKSPRX_OEM_KEY`
	`{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}`

VKSPTX_ID	Specifies the transmitter identifier. Every transmitter should have a unique identifier. Example of usage: `#define VKSP_ID {0,0,1}`

VKSPTX_OBSCURE_KEY	Specifies the 16-byte obscure key used for the VKSP algorithm. This parameter is used for obscuring and generating seeds. It is recommended to choose a random number for this parameter.
	
	Example of usage:
	
	`#define VKSP_OBSCURE_KEY`
	`{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}`

5 VKSP Transmitter Application Program Interface

After the driver is configured and compiled, you may use the application program interface (API) provided to integrate VKSP in the user application.

The following section details the VKSP API and the VKSP functions used to interface with the non-volatile memory block, the pseudo random number generation block, and the encryption block.

5.1 Application Program Interface (API)

5.1.1 VfnVkspTx_Init

Service Name	vfnVkspTx_Init

Syntax	`void vfnVkspTx_Init`
	`(`
	`  const VkspTx_ConfigType *sCfgPtr`
	`)`

Parameters in	Pointer to the selected structure. This structure contains the necessary data to initialize the VKSP module, taken from the configuration parameters.
Parameters out	None
Return value	None
Description	This routine initializes the VKSP module. It uses a global pointer to point to the selected constant structure. You can generate the structure by modifying the configuration parameters and using the variable sVksp_InitStruct. This function calls the PRNG interface the first time it is used.

5.1.2 VfnVkspTx_GenerateFrame

Service Name	vfnVkspTx_GenerateFrame

Syntax	`void vfnVkspTx_GenerateFrame`
	`(`
	`  VkspTx_CommandType Command,`
	`  VkspTx_MessageType *Message`
	`)`

Parameters in	Command: Command value to send to the receiver. Possible values go from 0 to 253. Values 254 and 255 are used only for learning frames. Message: This parameter is a pointer to the 16-byte array used to store the frame generated by this function.
Parameters out	None
Return value	None
Description	This function generates a command frame. The frame is created according to the protocol and is returned in the message pointer provided as parameter. This function is called by the main application before sending the frame to the receiver. This function calls the encryption interface and may call the memory interface if the value of the variable key has not been updated yet.

5.1.3 VfnVkspTx_PerformLearningSequence1

Service Name	vfnVkspTx_PerformLearningSequence1

Syntax	`void vfnVkspTx_PerformLearningSequence1`
	`(`
	`  VkspTx_MessageType *Message`
	`)`

Parameters in	Message: Pointer to a 16 byte-array used to store the learning sequence generated by this function.
Parameters out	None
Return value	None
Description	This function generates the first learning sequence frame. The user application should call this function and then send the generated learning frame. The learning sequence process must be executed once for every transmitter before initiating a normal communication with the receiver. The learning process consists in generating two learning sequence frames and transmit these to the receiver while the secure mode is enabled. If the driver is not initialized, this function does not perform any task.

5.1.4 VfnVkspTx_PerformLearningSequence2

Service Name	vfnVkspTx_PerformLearningSequence2

Syntax	`void vfnVkspTx_PerformLearningSequence2`
	`(`
	`  VkspTx_MessageType *Message`
	`)`

Parameters in	Message: Pointer to a 16 byte-array used to store the learning sequence generated by this function.
Parameters out	None
Return value	None
Description	This function implements the generation of the second learning sequence frame. This function verifies that the first learning sequence has already been generated. If this is not the case, the second learning sequence is not generated.

5.1.5 VfnVkspTx_UpdateVariableKey

Service Name	vfnVkspTx_UpdateVariableKey

Syntax	`void vfnVkspTx_UpdateVariableKey`
	`(`
	`  void`
	`)`

Parameters in	None
Parameters out	None
Return value	None
Description	This function updates the variable key value and must be called cyclically by a time interrupt function. Because the variable key must be different on every transmission, the developer must make sure the interrupt period is fast enough to afford two consecutive frames have different variable keys (at least one call between the generation of two consecutive frames). A common strategy is to set a slow interrupt rate while the microcontroller is in low-power mode (for example, 1 second) to reduce the power consumption, and increase the interrupt rate (for example 100ms) when the microcontroller wakes up. This allows the user to send several frames in a small time window.

5.2 Interfaces

Interfaces are functions that allow the VKSP Transmitter core to communicate with external blocks. Contrarily to the functions contained in the VKSP API, these interfaces are functions with bodies where code can be modified to fit the user application. The complexity of the code may vary depending on the user application and may go from a simple function call to data management necessary to fit the interface requirements.

5.2.1 VfnVkspTx_SaveSeeds

Service Name	vfnVkspTx_SaveSeeds

Syntax	`void vfnVkspTx_SaveSeeds`
	`(`
	`  VkspTx_SeedsType *Data`
	`)`

Parameters in	Data: Pointer to a buffer in RAM. Contents of this buffer are stored in non-volatile memory.
Parameters out	None
Return value	None
Description	This function must write 16 bytes into non-volatile memory. This function is called once in the life of the transmitter device. The address where this data is saved is defined by the user and is used as return value with vfnVkspTx_GetSeeds function.

5.2.2 VfnVkspTx_GetSeeds

Service Name	vfnVkspTx_GetSeeds

Syntax	`VkspTx_DataType* vfnVkspTx_GetSeeds`
	`(`
	`  void`
	`)`

Parameters in	None
Parameters out	None
Return value	It returns the address where the seeds are saved in non volatile memory.
Description	This function must return the starting address where the data is located in non-volatile memory.

5.2.3 VfnVkspTx_SetSetup

Service Name	vfnVkspTx_SetSetup

Syntax	`void vfnVkspTx_SetSetup`
	`(`
	`  void`
	`)`

Parameters in	None
Parameters out	None
Return value	None
Description	This function must program the value VKSPTX_INITIALIZED in non-volatile memory. This function is called once in the life of the transmitter application.

5.2.4 VfnVkspTx_GetSetup

Service Name	vfnVkspTx_GetSetup

Syntax	`VkspTx_SetupType vfnVkspTx_GetSetup`
	`(`
	`  void`
	`)`

Parameters in	None
Parameters out	None
Return value	Returns a value from non-volatile memory, possible values are VKSPTX_INITIALIZED and VKSPTX_UNINITIALIZED
Description	This function must return a value that indicates if the microcontroller was already initialized or if it is the first time it is powered on.

5.2.5 VfnVkspTx_SaveCounter

Service Name	vfnVkspTx_SaveCounter

Syntax	`void vfnVkspTx_SaveCounter`
	`(`
	`  void`
	`)`

Parameters in	None
Parameters out	None
Return value	None
Description	This function must increment the previous value of the 16-bits variable key (or counter) by one and save it in non-volatile memory.

5.2.6 VfnVkspTx_LoadCounter

Service Name	vfnVkspTx_LoadCounter

Syntax	`void vfnVkspTx_LoadCounter`
	`(`
	`  VkspTx_HCounterType *Counter`
	`)`

Parameters in	None
Parameters out	Counter: Pointer to the current value of the pointer
Return value	None
Description	This function must return the current value of the counter located in non-volatile memory.

5.2.7 VfnVkspTx_SaveLearnCounter

Service Name	vfnVkspTx_SaveLearnCounter

Syntax	`void vfnVkspTx_SaveLearnCounter`
	`(`
	`  VkspTx_LCounterType LCounter`
	`)`

Parameters in	LCounter: Value to be saved in the non-volatile location that holds the learning counter.
Parameters out	None
Return value	None
Description	This function must save the input parameter LCounter into non-volatile memory; it must also set the value of the normal counter to zero.

5.2.8 VfnVkspTx_LoadLearnCounter

Service Name	vfnVkspTx_LoadLearnCounter

Syntax	`void vfnVkspTx_LoadLearnCounter`
	`(`
	`  VkspTx_LCounterType *LCounter`
	`)`

Parameters in	None
Parameters out	LCounter: Value to be saved in the non-volatile location that holds the learning counter.
Return value	None
Description	This function must return the value of the learning counter in the location pointed by the argument.

5.2.9 VfnVkspTx_Encrypt

Service Name	vfnVkspTx_Encrypt

Syntax	`void vfnVkspTx_Encrypt`
	`(`
	`  const VkspTx_DataType *Data,`
	`  const VkspTx_DataType *Key,`
	`  VkspTx_DataType *Result`
	`)`

Parameters in	Data: Pointer to the 128 bits of raw data to be encrypted.
Key	Pointer to the 128-bit key used to encrypt data.
Parameters out	Result: Pointer to the 128-bit encrypted text result.
Return value	None
Description	This function performs a block encryption of 128 bits data/key and saves the result in a 128-bit result buffer.

5.2.10 VfnVkspTx_GeneratePRN

Service Name	vfnVkspTx_GeneratePRN

Syntax	`void vfnVkspTx_Encrypt`
	`(`
	`  VkspTx_DataType *Data,`
	`  VkspTx_DataType Size`
	`)`

Parameters in	Size: Number of random elements that are generated.
Parameters out	Data: Pointer to buffer in RAM where the generated pseudo random numbers are stored.
Return value	None
Description	This function generates [pseudo] random numbers. This function is used for the generation of the seeds. It is called once by vfnVkspTx_Init function. The parameter size indicates how many numbers must be generated. VKSP core requires only 16 numbers.

6 VKSP Transmitter Driver Implementation

This section explains how to set-up a RKE Transmitter application from scratch. The application is designed in a layered approach that facilitates the migration to different microcontrollers.

6.1 RKE Transmitter Overview

To build a RKE Transmitter application it is necessary to implement other modules aside from the VKSP transmitter core. You must understand what functionalities are managed by the VKSP transmitter core and what functionalities are not.

What VKSP transmitter core driver does:

- Reorganizes information to generate MACs.
- Manages the variable key
- Determines how to generate learning sequences

What VKSP transmitter core driver does not do:

- Encrypts
- Uses MCU resources directly
- Determines how the messages are sent by the physical layer

The present VKSP transmitter driver and the RKE application were designed using software architecture with a layered software design and to ease the migration to different platforms.

The software architecture used divides software into different layers. From a high-level perspective, the software can be described as follows:

- Hardware Abstraction Layer: Provides drivers to control MCU peripherals needed for the RKE application, such as timers, GPIO control, and KBI interrupts.
- Hardware Independent Layer: Provides drivers dependent on the electronic module used but independent on the hardware used. Examples of these drivers are the radio driver, the signal abstraction driver, and the VKSP driver.
- Services Layer: Drivers contained in this layer are often used by other layers. Timing control and power modes are two examples.

- Application Layer: This layer implements the application and initializes and integrates the rest of the drivers.

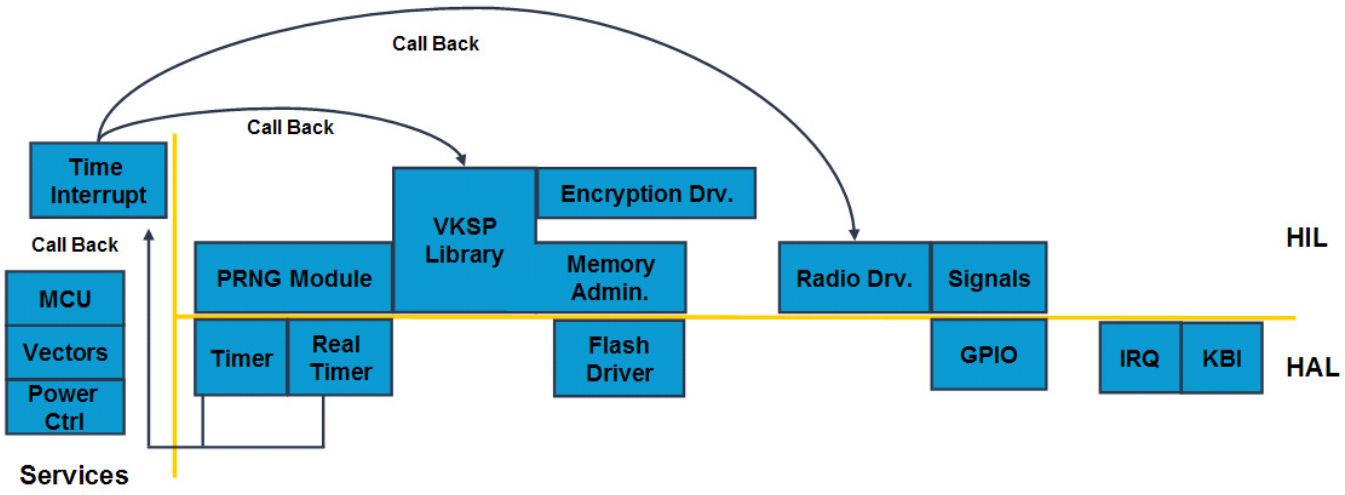

Figure 6. RKE Transmitter Layer Structure

6.2 Components Used

The transmitter device provides the following functionality:

- Reads inputs
- Implements low power modes
- Codifies messages
- Sends messages thru the RF link

Below is a brief description of the software components and their functionality.

6.2.1 HAL Drivers

- Real Time: This driver implements the software to control the real time clock or Interrupt peripheral. It implements a callback and an interrupt flag.
- Timer: This driver implements the software to control the TPM peripheral. It implements a callback and an interrupt flag. It is also possible to set the interrupt frequency based on the BUS frequency.
- Flash Driver: Driver implements control functions to write and erase flash memory. The functions are copied and executed from RAM.
- GPIO: Abstracts the microcontroller pins to virtual ports. It also implements a function to set the ports into low power modes.
- IRQ: This driver configures the IRQ peripheral to wake up the microcontroller when it is in SLEEP mode.
- KBI: This driver configures the KBI peripheral to wake up the microcontroller when it is in SLEEP mode.

6.2.2 HIL Drivers

- Signals: Abstracts the virtual ports to named signals from the electronic module. For example, it allows access to signals as buttons or LEDs. It uses the GPIO driver
- Radio: Handler that implements the functions to generate the messages received by the echo transceiver. It uses the TimingCtrl driver.
- PRNG: This driver generates pseudo random numbers. It uses uninitialized memory plus the inequity from the timer and real time clock sources to generate random numbers, depending on the selected configuration parameters
- AES: Implements the AES encryption module. This driver does not use any of the HAL drivers.
- VKSP Transmitter: Driver that implements the VKSP for the transmitter. Requires some files to interface with other modules, like the PRNG, flash, and the encryption module.

6.2.3 Services Drivers

- Common: Declares common types used by all other modules.
- MCU: Implements miscellaneous routines and macros like initialization, access to assembler instructions and public parameters like the bus frequency.
- TimingCtrl: Implements the usage of timer drivers and allows having two calls of different period using the same time base. The call period is configurable.
- Vectors: Defines the interrupts locations in program memory. This driver provides portability to the code.
- Low Power Ctrl: Controls the low-power modes in the microcontroller. It sets up the signals states and wake up signals.

6.2.4 Application

This layer integrates the HIL and Services drivers implementing the following features:

- Initializes the MCU, Signals, Memory, VKSP, Radio, and TimingCtrl drivers.
- Configures the callbacks for the TimingCtrl module.
- Enables/Disables the interrupts.
- Reads the state of the input signals and performs actions accordingly.
- Configures and sets the microcontroller up into stop mode.
- Reconfigures the microcontroller after a wake-up event.

6.2.5 Linker Parameters

Interrupts must be declared in the vectors module but allocation of variables and sections is done in the linker parameter file. VKSP library requires saving non-volatile data and must be reserved in this implementation space. The memory interface is open and implementation is up to you, but in this case the example provided uses one sector of memory to save the learning counter and the counter. That space must be reserved and not used for program code.

6.3 Memory Interface Implementation

Memory interface for VKSP is open and flexible so it can be adapted to user needs.

VKSP requires allocation for four non-volatile variables:

- Seeds (16 bytes)
- Setup (1 byte)
- Counter (2 bytes)
- Learning Counter (1 byte)

Seeds and setup are saved only once and never modified again. The counter and the learning counter require to be written over several times. The learning counter is overwritten each time a learning sequence is performed and the number of learning sequences that can be performed is limited. The counter is updated more often than the other variables each time there is a transmission and the lower counter has not overflowed there, is a write access to the non-volatile memory. If the average increment of the counter is T seconds, worst case per day of write accesses to the counter is:

$$\text{N access} = (\text{Seconds per Day}) / (2^{16} * T) \qquad \textit{Eqn. 1}$$

If the value of T is 0.70 seconds, there are approximately two accesses per day. In 20 years, there will be 14600 cycles. Make sure the memory used can afford this quantity of cycles.

A strategy must be implemented to overcome the physical limitations of the memory used. Based on the previous hypothetical case an S08 flash memory can afford such a stress. However, it is possible to implement a more complex memory management to increase the life of the peripheral.

The seeds and the setup variables can be allocated in any write-once memory space in flash memory. They are not overwritten and can be allocated anywhere in the same memory sector where code is located.

6.3.1 Simple Memory Model

The following section explains how to implement a method to save the values of the learning counter and the counter using one flash memory sector. This method provides reliability because it is simple. The drawback is that it has a limited life time and an erase action must be performed, adding execution time and power consumption, every time the values are updated. These drawbacks may not be an issue in most of the cases.

A one-byte space is reserved for the learning counter and a two-byte space is reserved for the counter. If the counter needs to be updated, learning counter and counter are first saved in RAM. Then, the memory sector is erased. Finally, the learning counter and the updated counter are programmed again. When a learning counter needs to be updated, the learning counter is saved in RAM, the memory sector is erased, and the updated learning counter and the cleared counter are programmed.

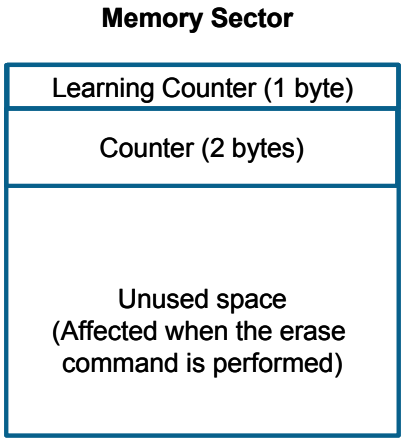

Figure 7. Simple Memory Model

6.3.2 Complex Memory Model

Because the counter is incremented by one at every update, it is not necessary to save the complete 2-byte counter. A flag that indicates the increment is sufficient. IIn one memory sector there is a one byte space to save the value of the learning counter, two bytes to save the value of the counter and the rest for flags that indicate the increments of the counters. Figure 8 describes the structure of this approach.

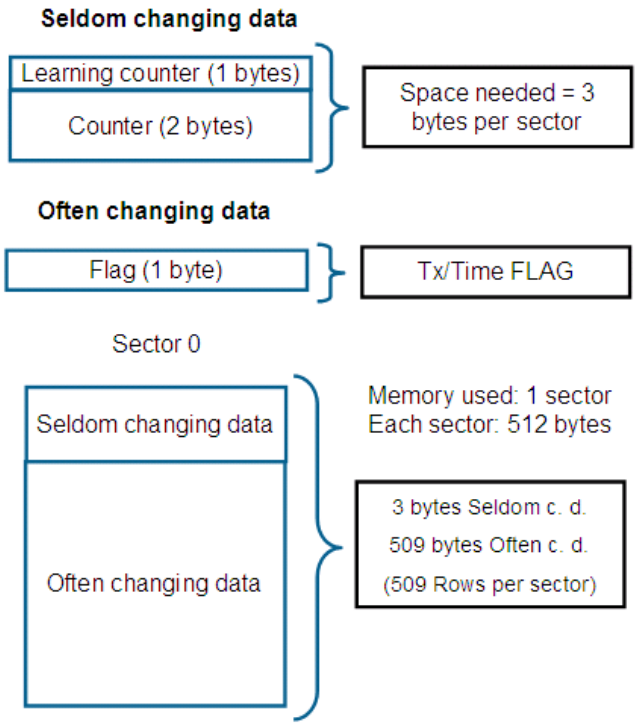

Figure 8. Complex Memory Model

In the complex memory model, the memory endurance is increased 510 times. Erase sequences in the transmitter are only done every 510 byte program sequences. This model is recommended for those applications where an extended life is required.

6.4 Encryption Interface Implementation

VKSP allows you to select the encryption protocol of your choice for the generation of MACs. The code for the call to the encryption driver must be located in the encryption interface. The interface requires the usage of a 128-bits block size encryption algorithm with a 128 bits key. Some encryption algorithms that do not comply with this requirement can be adapted so they can be used. Figure 9 depicts the adaptation of a hypothetical algorithm.

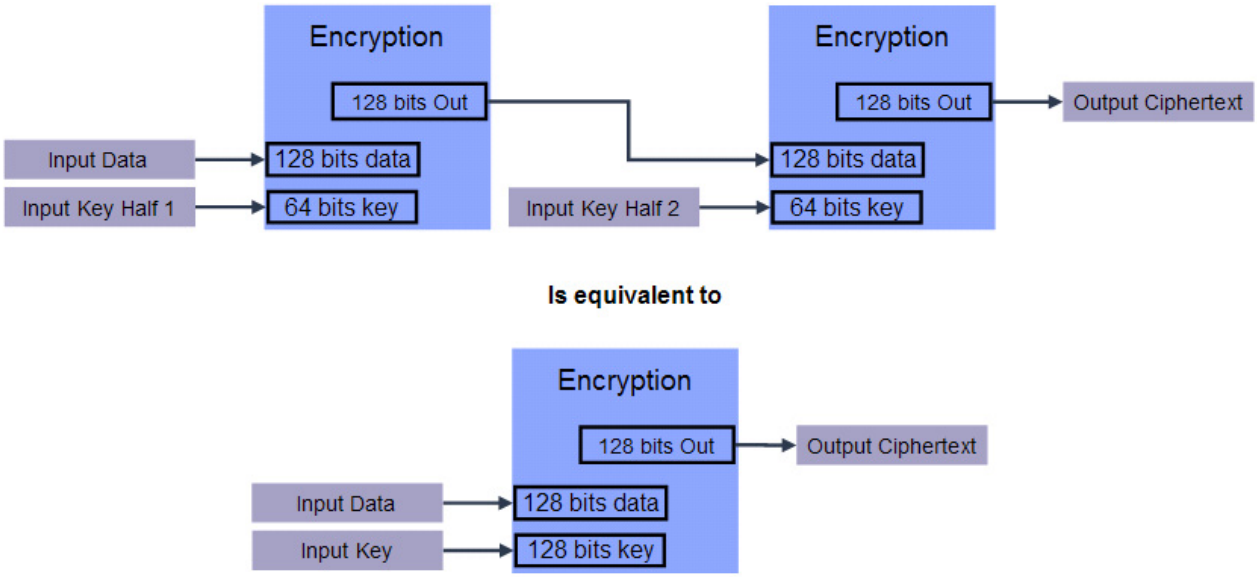

Figure 9. Encryption Adaptation to VKSP Requirements

In the application provided, the advanced encryption standard (AES) was chosen because it complies directly with the input/output requirements.

6.5 Pseudo Random Number Generation Interface Implementation

The PRNG interface generates random or pseudorandom numbers used as seeds for the VKSP core. Inside the PRNG interface, place a call to the function of your choice that does this.

The PRNG module provided with the example application uses uninitialized RAM locations and differences between the real time and timer events to produce pseudo-random values.

7 Step-by-Step Setup Guide of VKSP Transmitter Application

Before implementing the steps below, you must have:

- A driver to save and read values in non-volatile memory
- An encryption driver
- A pseudo random number generator

In the following steps, the term “if required” is used. This means it is possible to change the code provided in the sample application if the user requires it. For simplicity, this section does not explain how to implement the RF driver or other peripherals that have no direct relation with the VKSP driver.

1. Add the following files to your project: the vksp_tx.obj core file, the VKSP configuration files, (vksp_tx_cfg.h, vksp_tx_cfg.c) and the interface files (MemIf.c, CipherIf.c and PrngIf.c).
2. Modify the configuration parameters as needed (refer to Section 4, “Configuration of the VKSP Transmitter Driver”).
3. Modify code in the encryption interface if required. For example: Insert a call to your encryption driver.
4. Modify code in the memory interface if required
   — vfnVkspTx_SaveSeeds: Write code to save 16 bytes of data in non-volatile memory.
   — vfnVkspTx_GetSeeds: Write code to return the address to the first byte where the Seeds are located
   — vfnVkspTx_SetSetup: Write code to write into non-volatile memory the value VKSPTX_INITIALIZED.
   — vfnVkspTx_GetSetup: Write code to return the value of setup variable from non-volatile memory
   — vfnVkspTx_SaveCounter: Write code to increment the counter by one and save it in non-volatile memory. Size of this counter is 16 bits.
   — vfnVkspTx_LoadCounter: Write code to retrieve the value of the 16 bits counter. It is used only when the driver is initialized.
   — vfnVkspTx_SaveLearnCounter: Write code to save the learning counter in non-volatile memory and initialize the value of the normal counter to zero.
   — vfnVkspTx_LoadLearnCounter: Write code to retrieve the value from non-volatile memory of the learning counter.
5. Modify code in the pseudo random numbers generation interface if required. For example, insert within the body of the interface function which generator driver to call. The simplest solution is to gather those numbers from uninitialized RAM.
6. Setup vfnVkspTx_UpdateVariableKey call. This function must be called by a periodic interrupt. For example, while the microcontroller is in sleep mode, the real time module can wake up the microcontroller and call this function each second. When a button event is detected, increment the call rate to 100ms.

7. In the main application, initialize the driver that programs non-volatile memory before calling vfnVkspTx_Init.
8. In the main application, initialize VKSP by calling vfnVkspTx_Init.
9. To perform a learning sequence, call first vfnVkspTx_PerformLearningSequence1 and send the generated frame; then, call vfnVkspTx_PerformLearningSequence2 and send the generated frame.
10. To perform a normal transmission, call vfnVkspTx_GenerateFrame and send the frame.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: VKSPTXUG
Rev. 0
06/2008

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see http://www.freescale.com or contact your Freescale sales representative.

For information on Freescale’s Environmental Products program, go to http://www.freescale.com/epp.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.
© Freescale Semiconductor, Inc. 2008. All rights reserved.

freescale™
semiconductor