AN12554

Demo Application to Generate Red/Black Blobs Using CAAM and Encrypt/Decrypt Data

Rev. 0 — 22 August 2019 Application Note

Contents

1 Introduction

A blob is a cryptographic data structure that CAAM uses to protect data. It provides both confidentiality and integrity protection.

This document provides instructions and steps on how to set up and run a demo application to generate both red and black key blobs and use them to encrypt and decrypt data.

2 Overview

CAAM 's built-in blob protocol provides a method for protecting user-defined data across system power cycles. The data to be protected is encrypted so that it can be safely placed into non-volatile storage before the chip is powered down. Each time the blob protocol is used to protect data, a randomly generated key is used to encrypt the data. The random key is itself encrypted using a key encryption and is then stored along with the encrypted data. The encryption key is derived from the chip's master secret key so the encryption key can be recreated when the chip powers up again. The combination of encrypted key and encrypted data is called a blob. CAAM blobs differ depending on the data needed to be protected. Red blobs are intended for general data (left unencrypted when the blob is decapsulated). Black blob’s input during blob encapsulation is assumed to be a black key, and the output during blob decapsulation is either a black key that is written into memory, or an unencrypted key that is placed directly into a Key Register. A black key is an encrypted plain text key which is not power cycle safe, that's why it needs to be encapsulated into a blob. The black blob itself is exactly the same as a red blob, except that the BKEK derivation is different from red blobs. This prevents a black blob from being decapsulated as a red blob, which would leave the key exposed in memory.

This demo application uses secure memory blobs. The advantages of secure memory blobs over general memory blobs are:

- Secure memory blobs can be used only with secure memory, and all data must be encapsulated from, or decapsulated to, the same partition
- Secure memory blobs are subject to different secure memory access control restrictions than are general memory blobs
- Secure memory blob can be imported only into a partition with the same access permission settings as the partition from which the blob was exported

3 Components

Figure 1 illustrates the interaction between the components of the demo application.

To add kernel support for generating red/black blobs using CAAM and to encrypt/decrypt data using blobs, the key_blob kernel module must be added and enabled. To solve the requests received from the user space application, kernel module uses CAAM APIs and Secure Memory.

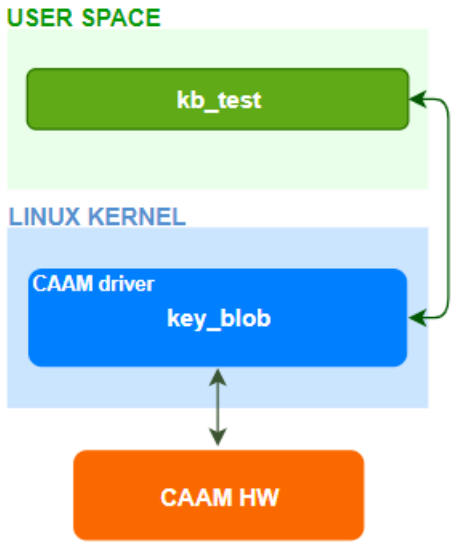

Figure 1. Overview of the demo application

CAAM provides some high-level cryptographic protocol operations like: memory slave interface that gives access to the Secure Memory and a DMA interface that allows CAAM to read/write data from external memory. The key_blob kernel module handles the allocation/deallocation of slots in secure memory page. and import/export of keys into/from memory blobs.

The kb_test user space application, executes the system call corresponding to the parameters received. If there is a request for encapsulating a key into a blob- It is used for encryption/decryption; the size of the key should be a multiple of 16 bytes. The demo application uses AES-ECB mode encryption which works with 128, 192 or 256-bit long key size. AES-ECB is a block cipher mode of operation and uses 16 bytes for the block size.

Before encapsulating a key into a blob, and encrypting a file with it; the key size should be adjusted in order to be 128/192/256 bit long. Also, the file size containing the data to be encrypted should be adjusted to be a multiple of 16 bytes. Otherwise, the kb_test application automatically adjusts the keyfile size and the size of the file that contains data to be encrypted.

4 Setup

To see the setup instructions and run the demo application Click here.

If you face any problems during execution of the application (for example: The blob generated is 0x0), check:

- All the kernel options are correctly enabled
- Demo application is in sync with the kernel version(Apply the right kernel app patch for your kernel version or update the kernel code according to the kernel version)

5 Usage

Encapsulating/Decapsulating a key into/from a blob

This section describes the steps required to encapsulate/decapsulate a key into/from a blob.

- Create a regular file with the desired key.
- Encapsulate/Decapsulate the key into/from a red/black blob:

```
$ ./kb_test encap <key_color> <key_file> <blob_file> //for encapsulation
$ ./kb_test decap <key_color> <blob_file> <key_file> //for decapsulation
```

<key_color> can be red or black

<key_file> is a regular file that contains the key to be encapsulated. This file should exist.

<blob_file> is the name of the file that will hold the blob

Encrypting/Decrypting a file using a blob

This section describes the steps required to encrypt/decrypt a file using a blob.

- Create a blob with the desired key and color.
- Encrypt/Decrypt a file using the blob:

```
$ kb_test encr <key_color> <blob_file> <input_file> <encrypted_file>       // for encryption
$ kb_test decr <key_color> <blob_file> <encrypted_file> <decrypted_file>  // for decryption
```

<key_color> can be red or black

<blob_file> is a regular file that contains the blob to be used for encryption/decryption. This file should exist.

<input_file> is a regular file that contains data to be encrypted_file. This file should exist.

<encrypted_file> is the name of the file that will contain the encrypted data

<decrypted_file> is the name of the file that will contain the decrypted data

6 Revision history

Revision number	Date	Substantive changes
0	08/2019	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP 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 NXP 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. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 22 August 2019
Document identifier: AN12554

arm