DOCA AES-GCM
Stopping

Alternative Datapath Options

DOCA AES-GCM Samples

Running the Samples

Samples

AES-GCM Encrypt

AES-GCM Decrypt
This guide provides instructions on building and developing applications that require data encryption and decryption using the AES-GCM algorithm.

**Introduction**

The library provides an API for executing AES-GCM operations on DOCA buffers, where the buffers reside in either local memory (i.e., within the same host) or host memory accessible by the DPU (remote memory). Using DOCA AES-GCM, complex encrypt/decrypt operations can be easily executed in an optimized, hardware-accelerated manner.

This document is intended for software developers wishing to accelerate their application's encrypt/decrypt operations.

**Prerequisites**

This library follows the architecture of a DOCA Core context, it is recommended to read the following sections before:

- [DOCA Core Execution Model](#)
- [DOCA Core Device](#)
- [DOCA Core Memory Subsystem](#)

**Environment**

DOCA AES-GCM-based applications can run either on the host machine or on the NVIDIA® BlueField® DPU target.

Encrypting/decrypting from the host to DPU and vice versa can only be run when the DPU is configured in DPU mode.
DOCA AES-GCM is a DOCA Core Context. This library leverages the DOCA Core architecture to expose asynchronous tasks/events that are offloaded to hardware.

AES-GCM can be used to encrypt/decrypt data as illustrated in the following diagrams:

- Encrypt/decrypt from local memory to local memory:

- Using the DPU to copy memory between the host and the DPU:

- Using the host to copy memory between the host and the DPU:
Objects

Device and Representor

The library requires a DOCA device to operate. The device is used to access memory and perform the actual encrypt/decrypt. See DOCA Core Device Discovery.

For the same BlueField DPU, it does not matter which device is used (i.e., PF/VF/SF) as all these devices utilize the same hardware component. If there are multiple DPUs, then it is possible to create a AES-GCM instance per DPU, providing each instance with a device from a different DPU.

To access memory that is not local (i.e., from the host to DPU or vice versa), the DPU side of the application must pick a device with an appropriate representor (see DOCA Core Device Representor Discovery). The device must stay valid as long as AES-GCM instance is not destroyed.

Memory Buffers

The encrypt/decrypt task, requires two DOCA buffers containing the destination and the source.

Depending on the allocation pattern of the buffers, consider the DOCA Core Inventory Types table.

To find what kind of memory is supported, refer to the following table.

Buffers must not be modified or read during the encrypt/decrypt operation.
Configuration Phase

To start using the library users must go through a configuration phase as described in DOCA Core Context Configuration Phase.

This section describes how to configure and start the context to allow execution of tasks and retrieval of events.

Configurations

The context can be configured to match the application use case.

To find if a configuration is supported or its min/max value, refer to Device Support.

Mandatory Configurations

These configurations must be set by the application before attempting to start the context:

- At least one task/event type must be configured. See configuration of Tasks and/or Events.
  - A device with appropriate support must be provided upon creation

Device Support

DOCA AES-GCM requires a device to operate. For picking a device, see DOCA Core Device Discovery.

As device capabilities may change in the future (see DOCA Core Device Support) it is recommended to select your device using the following method:

- doca_aes_gcm_cap_task_encrypt_is_supported
- doca_aes_gcm_cap_task_decrypt_is_supported
Some devices can allow different capabilities as follows:

- The maximum number of tasks
- The maximum buffer size
- The maximum supported number of elements in DOCA linked-list buffer
- The maximum initialization vector length
- Check if authentication tag of size 96-bit is supported
- Check if authentication tag of size 128-bit is supported
- Check if a given AES-GCM key type is supported

**Buffer Support**

Tasks support buffers with the following features:

<table>
<thead>
<tr>
<th>Buffer Type</th>
<th>Source Buffer</th>
<th>Destination Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local mmap buffer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mmap from PCIe export buffer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mmap from RDMA export buffer</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Linked list buffer</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**Execution Phase**

This section describes execution on the CPU using DOCA Core Progress Engine.

**Tasks**

DOCA AES-GCM exposes asynchronous tasks that leverage DPU hardware according to the DOCA Core architecture.
Encrypt Task

The encrypt task allows data encryption using buffers as described in Buffer Support.

Task Configuration

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td>doca_aes_gcm_task_encrypt_set_conf</td>
<td>doca_aes_gcm_cap_task_encrypt_is_supported</td>
</tr>
<tr>
<td>Number of tasks</td>
<td>doca_aes_gcm_task_encrypt_set_conf</td>
<td>doca_aes_gcm_cap_get_max_num_tasks</td>
</tr>
<tr>
<td>Maximal buffer size</td>
<td>-</td>
<td>doca_aes_gcm_cap_task_encrypt_get_max_buf_size</td>
</tr>
<tr>
<td>Maximum buffer list size</td>
<td>-</td>
<td>doca_aes_gcm_cap_task_encrypt_get_max_list_buf_num_elem</td>
</tr>
<tr>
<td>Maximum initialization vector length</td>
<td>-</td>
<td>doca_aes_gcm_cap_task_encrypt_get_max_iv_length</td>
</tr>
<tr>
<td>Enable authentication tag size</td>
<td>-</td>
<td>doca_aes_gcm_cap_task_encrypt_is_tag_96_supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>doca_aes_gcm_cap_task_encrypt_is_tag_128_supported</td>
</tr>
<tr>
<td>Enable key type</td>
<td>-</td>
<td>doca_aes_gcm_cap_task_encrypt_is_key_type_supported</td>
</tr>
</tbody>
</table>

Task Input

Common input as described in DOCA Core Task

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>source buffer</td>
<td>Buffer pointing to the memory to be encrypted</td>
<td>Only the data residing in the data segment is encrypted</td>
</tr>
<tr>
<td>destination buffer</td>
<td>Buffer pointing to where memory is encrypted to</td>
<td>The encrypted data is appended</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>key</td>
<td>Key to encrypt the data</td>
<td>Created by the function doca_aes_gcm_key_create. Users should use the same key to encrypt and decrypt the data</td>
</tr>
<tr>
<td>initializatio vector (IV)</td>
<td>Initialization vector to be used by the AES-GCM algorithm</td>
<td>Users should use the same IV to encrypt and decrypt the data.</td>
</tr>
<tr>
<td>initialization vector length</td>
<td>Initialization vector length that must be supplied for the AES-GCM algorithm</td>
<td>Represented in bytes, 0B-12B values are supported.</td>
</tr>
<tr>
<td>authentication tag size</td>
<td>Authentication tag size to be supplied for the AES-GCM algorithm. The tag is automatically calculated and appended to the result buffer.</td>
<td>Represented in bytes, only 12B and 16B values are supported.</td>
</tr>
<tr>
<td>additional authenticated data size</td>
<td>Additional authenticated data size to be supplied for the AES-GCM algorithm. This data, which should be present at the beginning of the source buffer, is will not encrypted but is authenticated.</td>
<td>Represented in bytes.</td>
</tr>
</tbody>
</table>

**Task Output**

Common output as described in DOCA Core Task.

**Task Completion Success**

After the task completes successfully, the following happens:
- The data from the source buffer is encrypted and written to the destination buffer
- The destination buffer data segment is extended to include the encrypted data

### Task Completion Failure

If the task fails midway:

- The context may enter stopping state if a fatal error occurs
- The source and destination `doca_buf` objects are not modified
- The destination buffer contents may be modified

### Task Limitations

- The operation is not atomic
- Once the task is submitted, the source and destination should not be read/written to
- Other limitations are described in DOCA Core Task

### Decrypt Task

The decrypt task allows data decryption. Using buffers as described in Buffer Support.

### Task Configuration

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_aes_gcm_task_decrypt_set_conf</code></td>
<td><code>doca_aes_gcm_cap_task_decrypt_is_supported</code></td>
</tr>
<tr>
<td>Description</td>
<td>API to Set Configuration</td>
<td>API to Query Support</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>--------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>Number of tasks</td>
<td>doca_aes_gcm_task_decrypt_set_conf</td>
<td>doca_aes_gcm_cap_get_max_num_tasks</td>
</tr>
<tr>
<td>Maximal buffer size</td>
<td>–</td>
<td>doca_aes_gcm_cap_task_decrypt_get_max_buf_size</td>
</tr>
<tr>
<td>Maximum buffer list size</td>
<td>–</td>
<td>doca_aes_gcm_cap_task_decrypt_get_max_list_buf_num_elem</td>
</tr>
<tr>
<td>Maximum initialization vector length</td>
<td>–</td>
<td>doca_aes_gcm_cap_task_decrypt_get_max_iv_length</td>
</tr>
<tr>
<td>Enable authentication tag size</td>
<td>–</td>
<td>doca_aes_gcm_cap_task_decrypt_is_tag_96_supported doca_aes_gcm_cap_task_decrypt_is_tag_128_supported</td>
</tr>
<tr>
<td>Enable key type</td>
<td>–</td>
<td>doca_aes_gcm_cap_task_decrypt_is_key_type_supported</td>
</tr>
</tbody>
</table>

**Task Input**

Common input as described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source buffer</td>
<td>Buffer pointing to the memory to be decrypted</td>
<td>Only the data residing in the data segment is decrypted</td>
</tr>
<tr>
<td>Destination buffer</td>
<td>Buffer pointing to where memory is decrypted to</td>
<td>The decrypted data is appended to the tail segment extending the data segment</td>
</tr>
<tr>
<td>Key</td>
<td>Key to decrypt the data</td>
<td>Created by the function doca_aes_gcm_key_create The user should use the same key to encrypt and decrypt the data</td>
</tr>
<tr>
<td>Initialization vector</td>
<td>Initialization vector to be used by the AES-GCM algorithm</td>
<td>The user should use the same IV to encrypt and</td>
</tr>
</tbody>
</table>

DOCA AES-GCM
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(IV)</td>
<td>decrypt the data</td>
<td></td>
</tr>
<tr>
<td>Initialization vector length</td>
<td>Initialization vector length that must be supplied for the AES-GCM algorithm</td>
<td>Represented in bytes, 0B-12B values are supported</td>
</tr>
<tr>
<td>Authentication tag size</td>
<td>Authentication tag size to be supplied for the AES-GCM algorithm. The tag, present at the end of the source buffer, is verified and is not present in the destination buffer.</td>
<td>Represented in bytes, only 12B and 16B values are supported</td>
</tr>
<tr>
<td>Additional authenticated data size</td>
<td>Additional authenticated data size to be supplied for the AES-GCM algorithm. This data, present at the beginning of the source buffer, is not encrypted but is authenticated.</td>
<td>Represented in bytes</td>
</tr>
</tbody>
</table>

**Task Output**

Common output as described in DOCA Core Task.

**Task Completion Success**

After the task completes successfully, the following happens:

- The data from the source buffer is decrypted and written to the destination buffer
- The destination buffer data segment is extended to include the decrypted data

**Task Completion Failure**

If the task fails midway:

- The context may enter stopping state if a fatal error occurs
- The source and destination doca_buf objects is not modified
• The destination buffer contents may be modified

Task Limitations

• The operation is not atomic

• Once the task is submitted, the source and destination should not be read/written to

• Other limitations are described in DOCA Core Task

Events

DOCA AES-GCM exposes asynchronous events to notify about changes that happen unexpectedly according to the DOCA Core architecture.

The only events AES-GCM exposes are common events as described in DOCA Core Event.

State Machine

The DOCA AES-GCM library follows the Context state machine as described in DOCA Core Context State Machine.

The following section describes moving states and what is allowed in each state.

Idle

In this state, it is expected that the application either:

• Destroys the context

• Starts the context
Allowed operations:

- Configuring the context according to Configurations
- Starting the context

It is possible to reach this state as follows:

<table>
<thead>
<tr>
<th>Previous State</th>
<th>Transition Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Create the context</td>
</tr>
<tr>
<td>Running</td>
<td>Call stop after making sure all tasks have been freed</td>
</tr>
<tr>
<td>Stopping</td>
<td>Call progress until all tasks are completed and freed</td>
</tr>
</tbody>
</table>

**Starting**

This state cannot be reached.

**Running**

In this state, it is expected that the application:

- Allocates and submits tasks
- Calls progress to complete tasks and/or receive events

Allowed operations:

- Allocating previously configured task
- Submitting a task
- Calling stop

It is possible to reach this state as follows:
<table>
<thead>
<tr>
<th>Previous State</th>
<th>Transition Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Call start after configuration</td>
</tr>
</tbody>
</table>

**Stopping**

In this state, it is expected that the application:

- Calls progress to complete all inflight tasks (tasks complete with failure)
- Frees any completed tasks

Allowed operations:

- Calling progress

It is possible to reach this state as follows:

<table>
<thead>
<tr>
<th>Previous State</th>
<th>Transition Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>Call progress and fatal error occurs</td>
</tr>
<tr>
<td>Running</td>
<td>Call stop without freeing all tasks</td>
</tr>
</tbody>
</table>

**Alternative Datapath Options**

DOCA AES-GCM only supports datapath on the CPU. See Execution Phase.

**DOCA AES-GCM Samples**

This section describes DOCA AES-GCM samples based on the DOCA AES-GCM library.

The samples in this section illustrate how to use the DOCA AES-GCM API to do the following:

- Encrypt contents of a buffer to another buffer
• Decrypt contents of a buffer to another buffer

Info

All the DOCA samples described in this section are governed under the BSD-3 software license agreement.

Running the Samples

1. Refer to the following documents:
   - NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software.
   - NVIDIA DOCA Troubleshooting Guide for any issue you may encounter with the installation, compilation, or execution of DOCA samples.

2. To build a given sample:

   ```
   cd /opt/mellanox/doca/samples/doca_aes_gcm/<sample_name>
   meson/tmp/build
   ninja -C/tmp/build
   ```

   Info

   The binary `doca_<sample_name>` is created under `/tmp/build/`.

3. Sample (e.g., `doca_aes_gcm_encrypt`) usage:

   ```
   Usage: doca_aes_gcm_encrypt [DOCA Flags] [Program Flags]
   ```
4. For additional information per sample, use the -h option:

```
/tmp/build/doca_<sample_name>-h
```

## Samples

### AES-GCM Encrypt

This sample illustrates how to encrypt data with AES-GCM.

The sample logic includes:

1. Locating DOCA device.

2. Initializing required DOCA Core structures.
3. Setting the AES-GCM encrypt tasks configuration.
4. Populating DOCA memory map with two relevant buffers.
5. Allocating element in DOCA buffer inventory for each buffer.
6. Creating DOCA AES-GCM key.
7. Allocating and initializing AES-GCM encrypt task.
8. Submitting AES-GCM encrypt task.
9. Retrieving AES-GCM encrypt task once it is done.
10. Checking task result.
11. Destroying all AES-GCM and DOCA Core structures.

Reference:

- `/opt/mellanox/doca/samples/doca_aes_gcm/aes_gcm_encrypt/aes_gcm_encrypt_sample.c`
- `/opt/mellanox/doca/samples/doca_aes_gcm/aes_gcm_encrypt/aes_gcm_encrypt_main.c`
- `/opt/mellanox/doca/samples/doca_aes_gcm/aes_gcm_encrypt/meson.build`

**AES-GCM Decrypt**

This sample illustrates how to decrypt data with AES-GCM.

The sample logic includes:

1. Locating DOCA device.
2. Initializing needed DOCA Core structures.
3. Setting the AES-GCM decrypt tasks configuration.
4. Populating DOCA memory map with two relevant buffers.
5. Allocating element in DOCA buffer inventory for each buffer.
6. Creating DOCA AES-GCM key.

7. Allocating and initializing AES-GCM decrypt task.

8. Submitting AES-GCM decrypt task.

9. Retrieving AES-GCM decrypt task once it is done.

10. Checking task result.

11. Destroying all AES-GCM and DOCA Core structures.

Reference:

- /opt/mellanox/doca/samples/doca_aes_gcm/aes_gcm_decrypt/aes_gcm_decrypt_sample.c
- /opt/mellanox/doca/samples/doca_aes_gcm/aes_gcm_decrypt/aes_gcm_decrypt_main.c
- /opt/mellanox/doca/samples/doca_aes_gcm/aes_gcm_decrypt/meson.build