Intel® QuickAssist Technology
英特尔快速辅助技术

API Programmer's Guide

API 程序员指南

Revision 011

版本 011

October 2021

2021 年 10 月

Document Number: 330864-011US

文件编号:330864-011US

Contents
内容

Figures
数字

Tables
桌子

Listing
列表

Revision History

修订历史

Document Number	Revision Number	Description	Revision Date
330684	011	In this release: • Updated Section 3.1 – added cpa_cy_ecsm2.h file reference as the API for SM2	October 2021
330684	010	• Updated Section 3.2.11 - TLS Key and MGF Mask Generation • Updated Section 3.4 - Using the SM2 API	June 2021
330684	009	• Updated cpaDcGenerateFooter Content	April 2021
330684	008	• Updated Section 1.3 Using this Document • Updated Section 3.2.2, Cipher AES-256 • Listing 7, Updated to use newer cipher function in place of occurrence 3DES • Updated Section 3.2.3, Hash with SHA-256 • Listing 14, Updated to use a newer hash function in place of occurrence of MD5 • Listing 18, Updated to use newer cipher function in place of occurrence 3DES • Updated Section 3.2.10 Chained Cipher and Hash using the Symmetric Data Plane API • Listing 41 and 42, updated to use newer cipher function in place of occurrence 3DES • Section 3.2.11.2, Setting CpaCyKeyGenTlsOpsData updated code	May 2020
330684	007	• Added Section 3.2.13, HKDF Use Case	February 2020
330684	006	• Added Section 4.5 Chained Hash and Stateless Compression	March 2019
330684	005	• Updated Section 3.1 Overview	December 2018
330684	004	• Updated Listing 13. Remove Cipher Session • Updated Section 4.0 Note • Updated Section 4.2 Sample – Stateful Data Compression • Updated Listing 56. Remove Stateful Session	June 2018
330684	003	• Added requirement that customers using Static or Dynamic Compression must decompress data and verify that it matches original source data.	February 2018
330684	002	• Updates to code samples.	September 2017

330684	001	• First public version of the document. Based on Intel® Confidential document number 442844-1.3 with the revision history of that document retained for reference purposes.	June 2014
文件号	**修订号**	**描述**	**修订日期**
330684	011	在此版本中: • 更新了第 3.1 节 – 添加了 cpa_cy_ecsm2.h 文件参考作为 sm2 的 API	2021 年 10 月
330684	010	• 更新了第 3.2.11 节- TLS 密钥和 MGF 掩码生成 • 更新了第 3.4 节-使用 SM2 API	2021 年 6 月
330684	009	• 更新了 cpaDcGenerateFooter 内容	2021 年 4 月
330684	008	• 使用本文件更新了第 1.3 节 • 更新了第 3.2.2 节，密码 AES-256 • 清单 7，更新后使用更新的密码函数代替出现的 3DES • 更新了第 3.2.3 节，哈希与 SHA-256 • 清单 14，更新后使用新的散列函数代替 MD5 • 清单 18，更新后使用更新的密码函数代替出现的 3DES • 更新了第 3.2.10 节“使用对称数据平面 API 的链式密码和哈希” • 清单 41 和 42，更新后使用更新的密码函数代替出现的 3DES • 3.2.11.2 部分，设置 CpaCyKeyGenTlsOpsData 更新代码	2020 年 5 月
330684	詹姆斯 • 邦德	• 增加了第 3.2.13 节，HKDF 用例	2020 年 2 月
330684	006	• 添加了第 4.5 节链式哈希和无状态压缩	2019 年 3 月
330684	005	• 更新了第 3.1 节概述	2018 年 12 月
330684	004	• 更新了清单 13。删除密码会话 • 更新了第 4.0 节注释 • 更新了第 4.2 节示例-有状态数据压缩 • 更新了清单 56。删除有状态会话	2018 年 6 月
330684	003	• 增加了使用静态或动态压缩的客户必须解压缩数据并验证其与原始源数据匹配的要求。	2018 年 2 月
330684	002	• 代码示例的更新。	2017 年 9 月
330684	001	• 该文档的第一个公开版本。基于英特尔机密文件编号 442844-1.3 保留该文件的修订历史以供参考。	2014 年 6 月

Document Number	Revision Number	Description	Revision Date
330684	011	In this release: • Updated Section 3.1 – added cpa_cy_ecsm2.h file reference as the API for SM2	October 2021
330684	1.3	• Added new API function, `cpaCySymSessionCtxGetDynamicSize()`, to Section 3.2.2.2, plus other minor updates.	May 2014
330684	1.2	• Updated Listing 26, 27, and 30 in Section 3.2	January 2014
330684	1.1	Updates for stateless data compression samples: • Updated Section 4.3 • Added Section 4.4	March 2013
330684	1.0	• Initial release of the document	September 2012
文件号	修订号	描述	修订日期
330684	011	在此版本中： • 更新了第 3.1 节 – 添加了 cpa_cy_ecsm2.h 文件参考作为 sm2 的 API	2021 年 10 月
330684	1.3	• 向 3.2.2.2 部分添加了新的 API 函数 cpaCySymSessionCtxGetDynamicSize()，以及其他小的更新。	2014 年 5 月
330684	1.2	• 更新了第 3.2 节中的清单 26、27 和 30	2014 年 1 月
330684	1.1	无状态数据压缩示例的更新： • 更新了第 4.3 节 • 增加了第 4.4 节	2013 年 3 月
330684	1.0	• 文件的首次发布	2012 年 9 月

§

1 Introduction

2 介绍

This API programmer's guide describes the sample code that demonstrates how to use the Intel® QuickAssist Technology (Intel® QAT) APIs.

本 API 程序员指南描述了演示如何使用英特尔快速辅助技术(英特尔 QAT)API 的示例代码。

2.1 Intended Audience

2.2 目标受众

This document is intended to be used by software engineers who wish to develop application software that uses the Intel® QAT APIs to accelerate the supported workloads and/or services.

本文档面向希望开发使用英特尔 QAT APIs 来加速支持的工作负载和/或服务的应用软件的软件工程师。

2.3 Related Documents and References

2.4 相关文件和参考资料

Table 1. Related Documents and References

表 1。相关文件和参考资料

Document	Document Number /Location
Intel® QuickAssist Technology Cryptographic API Reference Manual	330685
Intel® QuickAssist Technology Data Compression API Reference Manual	330686
Intel® QuickAssist Technology Performance Optimization Guide	330687
Intel® QuickAssist Technology Programmer's Guide for Linux* - Hardware Version 1.7	336210
Intel® Communications Chipset 8925 to 8955 Series Software Programmer's Guide	330751
Intel® Communications Chipset 8900 to 8920 Series Software Programmer's Guide	330753
Intel Atom® Processor C2000 Product Family for Communications Infrastructure Software Programmer's Guide	330755
Intel® QuickAssist Technology Software for Linux* Release Notes	336211
NIST publication SP800-38C Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality	https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38c.pdf

NIST publication SP800-38D Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC	https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf
文件	**文件编号/位置**
英特尔快速辅助技术加密 API 参考手册	330685
英特尔快速辅助技术数据压缩 API 参考手册	330686
英特尔快速辅助技术性能优化指南	330687
面向 Linux*的英特尔快速辅助技术程序员指南-硬件版本 1.7	336210
英特尔通信芯片组 8925 至 8955 系列软件程序员指南	330751
英特尔通信芯片组 8900 至 8920 系列软件程序员指南	330753
面向通信基础设施软件程序员指南的英特尔凌动处理器 C2000 产品家族	330755
面向 Linux*的英特尔快速辅助技术软件发行说明	336211
NIST 出版物 SP800-38C 块密码操作模式的推荐标准:认证和保密的 CCM 模式	https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38c.pdf
NIST 出版物 SP800-38D 分组密码操作模式的推荐标准:伽罗瓦/计数器模式(GCM)和 GMAC	https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf

Document	Document Number /Location
NIST SP 800-90, March 2007 Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)	https://csrc.nist.gov/publications/detail/sp/800-90/revised/archive/2007-03-14
GZIP file format specification v4.3	RFC 1952
GZIP file format specification v3.3	RFC 1950
The Transport Layer Security (TLS) Protocol Version 1.0	RFC 2246
The Transport Layer Security (TLS) Protocol Version 1.1	RFC 4346
The Transport Layer Security (TLS) Protocol Version 1.2	RFC 5246
The Secure Sockets Layer (SSL) Protocol Version 3.0	RFC 6106
文件	**文件编号/位置**
NIST SP 800-90，2007 年 3 月 使用确定性随机位发生器产生随机数的推荐标准(修订版)	https://csrc.nist.gov/publications/detail/sp/800-90/revised/archive/2007-03-14
GZIP 文件格式规范 4.3 版	RFC 1952
GZIP 文件格式规范 3.3 版	RFC 1950
传输层安全(TLS)协议版本 1.0	RFC 2246
传输层安全(TLS)协议版本 1.1	RFC 4346
传输层安全(TLS)协议版本 1.2	RFC 5246
安全套接字层(SSL)协议版本 3.0	RFC 6106

2.5 Using This Document

2.6 使用此文档

This document is structured as follows:

本文件的结构如下:

- Section 2 Base API and API Conventions describes aspects common to all Intel® QuickAssist Technology APIs
- Section 2 Base API and API Conventions 描述所有英特尔 QuickAssist 技术 API 共有的方面
- Section 3 Intel® QuickAssist Technology Cryptographic API describes the Intel® QuickAssist Technology Cryptographic API
- Section 3 Intel® QuickAssist Technology Cryptographic API 描述英特尔快速辅助技术加密 API
- Section 4 Intel® QuickAssist Technology Data Compression API describes the Intel®

QuickAssist Technology Data Compression API

- Section 4 Intel® QuickAssist Technology Data Compression API 描述英特尔快速辅助技术数据压缩 API

Code for all the examples in this document is contained in the software package and, after installation, can be found in a sub-directory of the following directory:
本文档中所有示例的代码都包含在软件包中，安装后可以在以下目录的子目录中找到:

`quickassist/lookaside/access_layer/src/sample_code/functional`

快速辅助/旁视/访问层/src/样本代码/功能

Refer to Table 2 for a list of terms and acronyms used in this manual.
涉及 Table 2

2.7 Terminology

2.8 术语

Table 2. Terminology

表二。术语

Term	Description
AAD	Additional Authenticated Data
AES	Advanced Encryption Standard
API	Application Programming Interface
CBC	Cipher Block Chaining
CCM	Counter mode with Cipher-block chaining Message authentication code
学期	**描述**
模拟式音乐母盘	附加认证数据
俄歇电子能谱	高级加密标准
应用程序接口	应用程序界面
加拿大广播公司	密码分组链接
反干扰措施（counter-counter measures 的缩写）	具有密码块链接消息认证码的计数器模式

Term	Description
CPM	Content Processing Module
CY	Cryptographic
DC	Data Compression
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECDH	Elliptic Diffie-Hellman
EC	Elliptic Curve
ESP	Encapsulating Security Payload
GCD	Greatest Common Divisor
GCM	Galois Counter Mode
HKDF	HMAC Key Derivation Function
HMAC	Hashed Message Authenticate Code
ICV	Integrity Check Value
IPSec	Internet Protocol Security
MAC	Message Authentication Code
NRBG	Non-Deterministic Random Bit Generator
PKE	Public Key Encryption
PV	Pubic Value
Intel® QAT	Intel® QuickAssist Technology
RBG	Random Bit Generation
RSA	A public-key encryption algorithm created by Rivest, Shamir, and Adleman
SSL	Secure Sockets Layer
TLS	Transport Layer Security (SSL successor)
学期	**描述**
每千成本（cost per mille）	内容处理模块
塞浦路斯	关于暗号的
直流电	数据压缩
确定性随机比特生成器	确定性随机位生成器

目录系统代理 (Directory System Agent)	数字签名算法
密钥交换	椭圆迪菲-赫尔曼
东部中央邮（政）区	椭圆曲线
电动选择型	封装安全负载
GCD	最大公约数
最大公测度	伽罗瓦计数器模式
HKDF	HMAC 密钥导出函数
HMAC	散列消息验证码
回盲瓣 (ileocecal valve)	完整性检查值
安全协议	互联网协议安全性
测量与控制 (Measurement and Control)	消息认证代码
NRBG	非确定性随机位生成器
无钥匙进入	公钥加密
产品鉴定 (Production Validation)	公共价值
英特尔 QAT	英特尔快速辅助技术
《鲁斯・巴德・金斯伯格》	随机位生成
南非共和国 (Republic of South Africa)	Rivest、Shamir 和 Adleman 创建的公钥加密算法
加密套接字协议层	安全套接字层
坦克激光瞄准镜（Tank Laser-Sight 的缩写）	传输层安全性(SSL 后继协议)

§

3 Base API and API Conventions
4 基本API和API约定

This chapter describes aspects common to all Intel® QAT Technology APIs, starting with the base API and followed by conventions.

本章描述了所有英特尔QAT技术API共有的方面，从基本API开始，然后是约定。

4.1 Intel® QAT Technology Base API
4.2 英特尔QAT技术基础API

The Base API is a top-level API definition for Intel® QAT Technology. It contains structures, data types, and definitions that are common across the interface.

基本API是英特尔QAT技术的顶级API定义。它包含跨接口通用的结构、数据类型和定义。

2.1.1 Data Buffer Models
2.1.2 数据缓冲模型

Data buffers are passed across the API interface in one of the following formats:

数据缓冲区以下列格式之一通过API接口传递：

- Flat Buffers represent a single region of physically contiguous memory and are described in detail in Section 2.1.1.1, Flat Buffers.
- 平面缓冲区代表物理上连续内存的单个区域，在第节中有详细描述2.1.1.1Flat Buffers
- Scatter-Gather Lists are essentially an array of flat buffers, for cases where the memory is not all physically contiguous. These are described in detail in Section 2.1.1.2, Scatter-Gather Lists.
- 分散-聚集列表本质上是一个平面缓冲区数组，用于内存在物理上不连续的情况。这些将在第节中详细描述 2.1.1.2Scatter-Gather Lists

2.1.1.1 Flat Buffers
2.1.1.2 平面缓冲器

Flat buffers are represented by the type `CpaFlatBuffer`, defined in the file `cpa.h`. It consists of two fields:

平面缓冲区由cpa.h文件中定义的CpaFlatBuffer类型表示。它由两个字段组成：

- Data pointer pData: points to the start address of the data or payload. The data pointer is a virtual address; however, the actual data pointed to is required to be in contiguous and DMAable physical memory. This buffer type is typically used when simple, unchained buffers are needed.

- 数据指针 pData:指向数据或有效载荷的起始地址。数据指针是虚拟地址；但是，指向的实际数据需要在连续的、可 DMA 的物理内存中。这种类型的缓冲区通常在需要简单的非链式缓冲区时使用。
- Length of this buffer: dataLenInBytes specified in bytes.
- 该缓冲区的长度:以字节为单位指定的 dataLenInBytes。

For data plane APIs (`cpa_sym_dp.h` and `cpa_dc_dp.h`), a flat buffer is represented by the type `CpaPhysFlatBuffer`, also defined in `cpa.h`. This is similar to the `CpaFlatBuffer` structure; the difference is that, in this case, the data pointer, `bufferPhysAddr`, is a physical address rather than a virtual address.

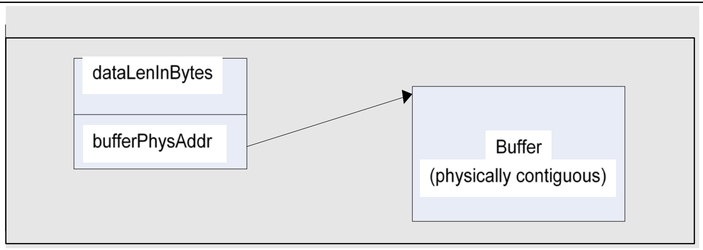
对于数据平面 API(CPA _ sym _ DP . h 和 cpa_dc_dp.h)，平面缓冲区由 CpaPhysFlatBuffer 类型表示，也是在 cpa.h 中定义的，这类似于 CpaFlatBuffer 结构；区别在于，在这种情况下，数据指针 bufferPhysAddr 是物理地址，而不是虚拟地址。

Figure 1 shows the layout of a flat buffer.

Figure 1 显示了平面缓冲区的布局。

Figure 1. Flat Buffer Diagram

图一。平面缓冲图



2.1.1.3 Scatter-Gather Lists
2.1.1.4 分散-聚集列表

A scatter-gather list is defined by the type CpaBufferList, also defined in the file cpa.h. This buffer structure is typically used where more than one flat buffer can be provided to a particular API. The buffer list contains four fields, as follows:

分散-聚集列表由 CpaBufferList 类型定义，也在 cpa.h 文件中定义。这种缓冲区结构通常用于可以向特定 API 提供多个平面缓冲区的情况。缓冲列表包含四个字段，如下所示:

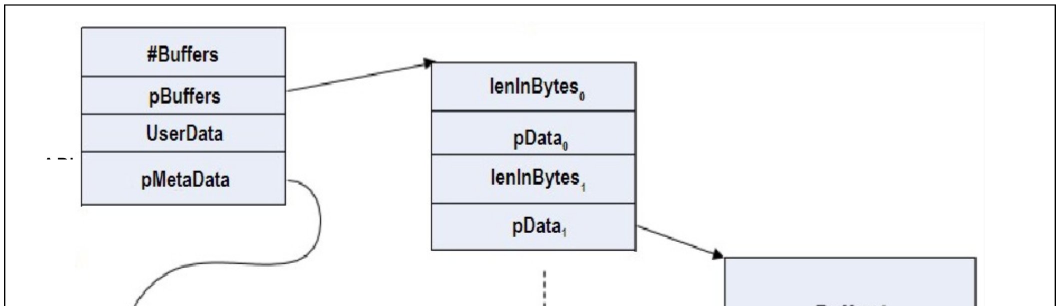
- The number of buffers in the list.
- 列表中缓冲区的数量。
- Pointer to an unbounded array of flat buffers.
- 指向平面缓冲区的无界数组的指针。
- User Data: an opaque field; is not read or modified internally by the API.
- 用户数据:一个不透明的领域；不是由 API 内部读取或修改的。
- This field could be used to provide a pointer back into an application data structure, providing the context of the call.
- 该字段可用于提供一个指向应用数据结构的指针，提供调用的上下文。
- Pointer to metadata required by the API:
- 指向 API 所需元数据的指针:
  - The metadata is required for internal use by the API. The memory for this buffer needs to be allocated by the client as contiguous data. The size of this metadata buffer is obtained by calling cpaCyBufferListGetMetaSize for crypto, cpaBufferLists, and cpaDcBufferListGetMetaSize for data compression.
  - API 内部使用时需要元数据。客户端需要将该缓冲区的内存作为连续数据进行分配。此元数据缓冲区的大小是通过调用用于加密的 cpaCyBufferListGetMetaSize、用于数据压缩的 cpaBufferLists 和 cpaDcBufferListGetMetaSize 获得的。
  - The memory required to hold the CpaBufferList structure and the array of flat buffers is not required to be physically contiguous. However, the flat buffer data pointers and the metadata pointer are required to reference physically contiguous DMAable memory.
  - 保存 CpaBufferList 结构和平面缓冲器阵列所需的存储器不需要在物理上是连续的。但是，需要平面缓冲区数据指针和元数据指针来引用物理上连续的可 DMA 内存。
  - There is a performance impact when using scatter-gather lists instead of flat buffers. Refer to Table 1, Intel® QAT Performance Optimization Guide for additional information.
  - 当使用分散-聚集列表而不是平面缓冲区时，会对性能产生影响。涉及 Table 1

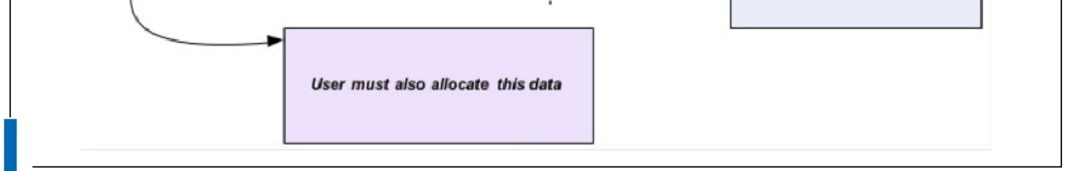
Figure 2 shows a graphical representation of a scatter-gather buffer list.

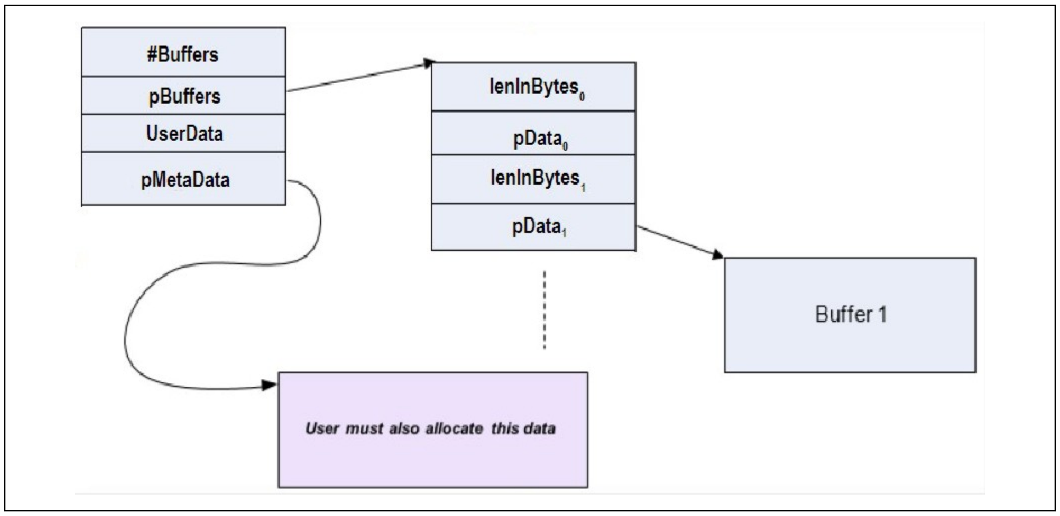
Figure 2 显示了分散-聚集缓冲区列表的图形表示。

Figure 2. Scatter-Gather List Diagram

图二。分散-聚集列表图







For data plane APIs (cpa_sym_dp.h and cpa_dc_dp.h) a region of memory that is not physically contiguous is described using the CpaPhysBufferList structure. This is similar to the CpaBufferList structure; the difference, in this case, the individual flat buffers are represented using physical rather than virtual addresses.

对于数据层 API(CPA _ sym _ DP . h 和 cpa_dc_dp.h),物理上不连续的内存区域使用 CpaPhysBufferList 结构进行描述。这类似于 CpaBufferList 结构；区别在于，在这种情况下，各个平面缓冲区使用物理地址而不是虚拟地址来表示。

4.3 Intel® QuickAssist Technology API Conventions

4.4 英特尔快速辅助技术 API 惯例

2.2.1 Instance Discovery

2.2.2 实例发现

Intel® QAT API supports multiple instances. An instance represents a "channel" to a specific hardware accelerator. Multiple instances can access the same hardware accelerator (that is, the relationship between instances and a hardware accelerator is N:1). The instance is identified using the `CpaInstanceHandle` handle type. This handle type represents a specific instance within the system and is passed as a parameter to all API functions that operate on instances.

英特尔 QAT API 支持多个实例。一个实例代表一个特定硬件加速器的“通道”。多个实例可以访问同一个硬件加速器(也就是说，实例和硬件加速器之间的关系是 N:1)。使用 CpaInstanceHandle 句柄类型标识该实例。该句柄类型表示系统中的特定实例，并作为参数传递给所有对实例进行操作的 API 函数。

Instance discovery is achieved through service-specific API invocations. This section describes the instance discovery for data compression (`dc`); however, the flow of the calls is similar for the cryptographic service.
实例发现是通过特定于服务的 API 调用实现的。本节描述数据压缩(dc)的实例发现；然而，对于加密服务，调用的流程是相似的。

Listing 1. Getting an Instance
清单 1。获取实例

```
void sampleDcGetInstance (CpaInstanceHandle *pDcInstHandle)
{
    CpaInstanceHandle dcInstHandles[MAX_INSTANCES];
    Cpa16U numInstances = 0;
    CpaStatus status = CPA_STATUS_SUCCESS;

    *pDcInstHandle = NULL;

    status = cpaDcGetNumInstances(&numInstances);
    if ((status == CPA_STATUS_SUCCESS) && (numInstances > 0)) {
        status = cpaDcGetInstances(MAX_INSTANCES, dcInstHandles);
        if (status == CPA_STATUS_SUCCESS) {
            *pDcInstHandle = dcInstHandles[0];
        }
    }

    if (0 == numInstances) {
        PRINT_ERR("No instances found for 'SSL'\n");
        PRINT_ERR("Please check your section names in the config
file.\n");
        PRINT_ERR("Also make sure to use config file version 2.\n");
    }
}
```

```
void sampleDcGetInstance (CpaInstanceHandle *pDcInstHandle)
{
    CpaInstanceHandle dcInstHandles[MAX_INSTANCES]; Cpa16U
    numInstances = 0;
    CpaStatus status = CPA_STATUS_SUCCESS;

    *pDcInstHandle = NULL;

    status = cpaDcGetNumInstances(&numInstances);
    if ((status == CPA_STATUS_SUCCESS) && (numInstances > 0)) { status =
        cpaDcGetInstances(MAX_INSTANCES, dcInstHandles); if (status ==
        CPA_STATUS_SUCCESS) {
            *pDcInstHandle = dcInstHandles[0];
        }
    }

    if (0 == numInstances) {
        PRINT_ERR("No instances found for 'SSL'\n");
        PRINT_ERR("Please check your section names in the config file.\n");
        PRINT_ERR("Also make sure to use config file version 2.\n");
    }
}
```

In this example, the number of `dc` instances available to the Application is queried via the `cpaDcGetNumInstances` call. The Application obtains the instance handle of the first instance.

在此示例中，通过 cpaDcGetNumInstances 调用来查询应用程序可用的 dc 实例的数量。应用程序获得第一个实例的实例句柄。

The next example shows the Application querying the capabilities of the data compression implementation, and verifying the required functionality is present. Each service implementation exposes the capabilities that have been implemented and are available. Capabilities include algorithms, common features, and limits to variables.
下一个示例显示了应用程序查询数据压缩实现的功能，并验证所需的功能是否存在。每个服务实现都公开了已经实现的和可用的功能。功能包括算法、公共特性和对变量的限制。

Each service has a unique capability matrix, and each implementation identifies and describes its particular implementation through its capability's API.
每个服务都有一个独特的功能矩阵，每个实现都通过其功能的 API 来标识和描述其特定的实现。

Listing 2. Querying and Starting an Instance
清单 2。查询和启动实例

```
status = cpaDcQueryCapabilities(dcInstHandle,
&cap); if (status != CPA_STATUS_SUCCESS) {
    return status;
}
if (!cap.statelessDeflateCompression || !
cap.statefulDeflateDecompression
||
    !cap.checksumCRC32 || !cap.dynamicHuffman) {
    PRINT_ERR("Error: Unsupported functionality\
    n"); return CPA_STATUS_FAIL;
}
/*
 * Set the address translation function for the instance
 */
status = cpaDcSetAddressTranslation(dcInstHandle,
sampleVirtToPhys); if (CPA_STATUS_SUCCESS == status) {
    /* Start DataCompression component
     * In this example we are performing static compression so
     * an intermediate buffer is not required
     */ PRINT_DBG("cpaDcStartInstance\n");
    status = cpaDcStartInstance(dcInstHandle, 0, NULL);
```

```
status = cpaDcQueryCapabilities(dcInstHandle, &cap); if
(status != CPA_STATUS_SUCCESS) {
    return status;
}
if (!cap.statelessDeflateCompression || !cap.statefulDeflateDecompression
||
    !cap.checksumCRC32 || !cap.dynamicHuffman) {
    PRINT_ERR("Error: Unsupported functionality\n"); return
    CPA_STATUS_FAIL;
}
/*
 * Set the address translation function for the instance
 */
status = cpaDcSetAddressTranslation(dcInstHandle, sampleVirtToPhys); if
(CPA_STATUS_SUCCESS == status) {
    /* Start DataCompression component
     * In this example we are performing static compression so
     * an intermediate buffer is not required
     */ PRINT_DBG("cpaDcStartInstance\n");
    status = cpaDcStartInstance(dcInstHandle, 0, NULL);
```

In the example, the application requires stateless deflate compression with dynamic Huffman encoding and stateful decompression with support for CRC32 checksums. The example also sets the address translation function for the instance. The specified function is used by the API to perform any required translation of a virtual address to a physical address. Finally, the instance is started.
在这个例子中，应用程序需要使用动态霍夫曼编码的无状态 deflate 压缩和支持 CRC32 校验和的有状态解压缩。该示例还为实例设置了地址转换函数。API 使用指定的函数将虚拟地址转换为物理地址。最后，启动实例。

2.2.3 Modes of Operation
2.2.4 操作模式

The Intel® QAT API supports both synchronous and asynchronous modes of operation. For optimal performance, the Application should be capable of submitting multiple outstanding requests to the acceleration engines. Submitting multiple outstanding requests minimizes the processing latency on the acceleration engines. This can be done by submitting requests asynchronously or by submitting requests in synchronous mode using multi-threading in the Application.

英特尔 QAT API 支持同步和异步操作模式。为了获得最佳性能，应用程序应该能够向加速引擎提交多个未完成的请求。提交多个未完成的请求可以最大限度地减少加速引擎的处理延迟。这可以通过异步提交请求或在应用程序中使用多线程以同步模式提交请求来实现。

Developers can select the mode of operation that best aligns with their Application and system architecture.
开发人员可以选择最符合其应用和系统架构的操作模式。

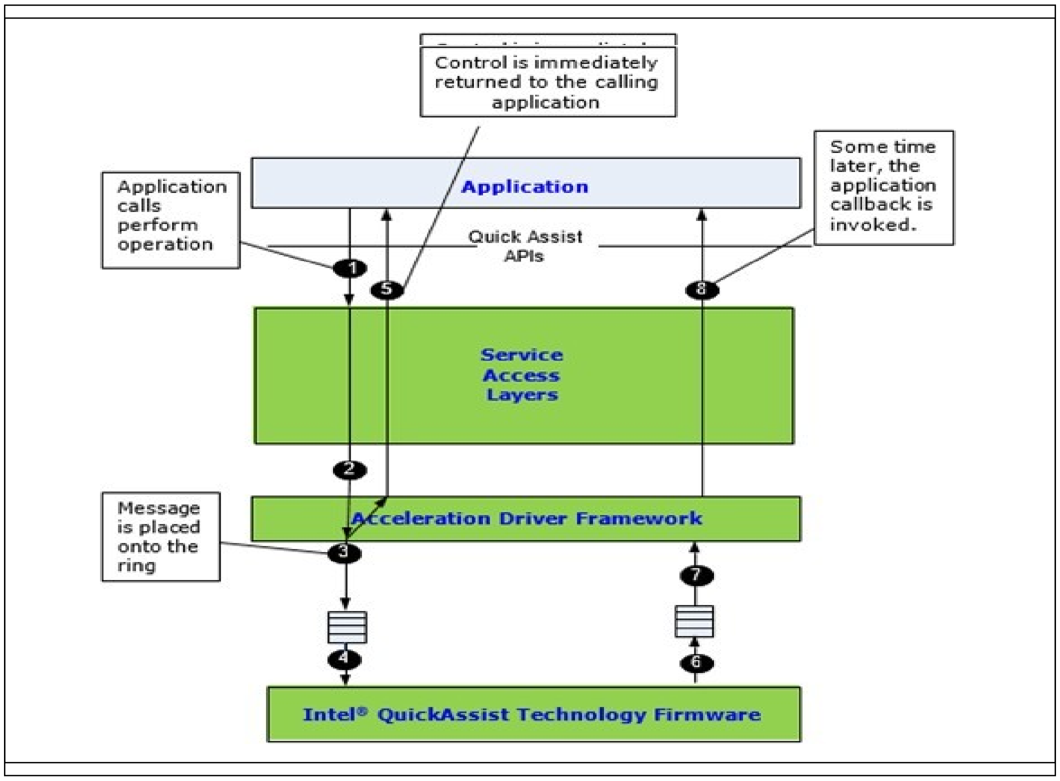
2.2.4.1 Asynchronous Operation
2.2.4.2 异步操作

To invoke the API asynchronously, the user supplies a callback function to the API, as shown in Figure 3. Control returns to the client once the request has been sent to the hardware accelerator, and the callback is invoked when the engine completes the

为了异步调用 API，用户向 API 提供一个回调函数，如 Figure 3

operation. The mechanism used to invoke the callback is implementation-dependent. For some implementations, the callback is invoked as part of an interrupt handler bottom half. For other implementations, the callback is invoked in the context of a polling thread. In this case, the user application is responsible for creating and scheduling this polling thread. Refer to Table 1 for the implementation of specific documentation for more details.
操作。用于调用回调的机制是依赖于实现的。对于某些实现，回调作为中断处理程序下半部分的一部分被调用。对于其他实现，回调在轮询线程的上下文中调用。在这种情况下，用户应用程序负责创建和调度这个轮询线程。涉及 Table 1

Figure 3. Asynchronous Operation
图 3。异步操作


Control is immediately returned to the calling application
Some time later, the application callback is invoked.
Application calls perform operation
Application
Quick Assist APIs
1
5
8
Service Access Layers
2
Message is placed onto the ring
Acceleration Driver Framework
3
7
4
6
Intel® QuickAssist Technology Firmware

2.2.4.3 Synchronous Operation

2.2.4.4 同步运行

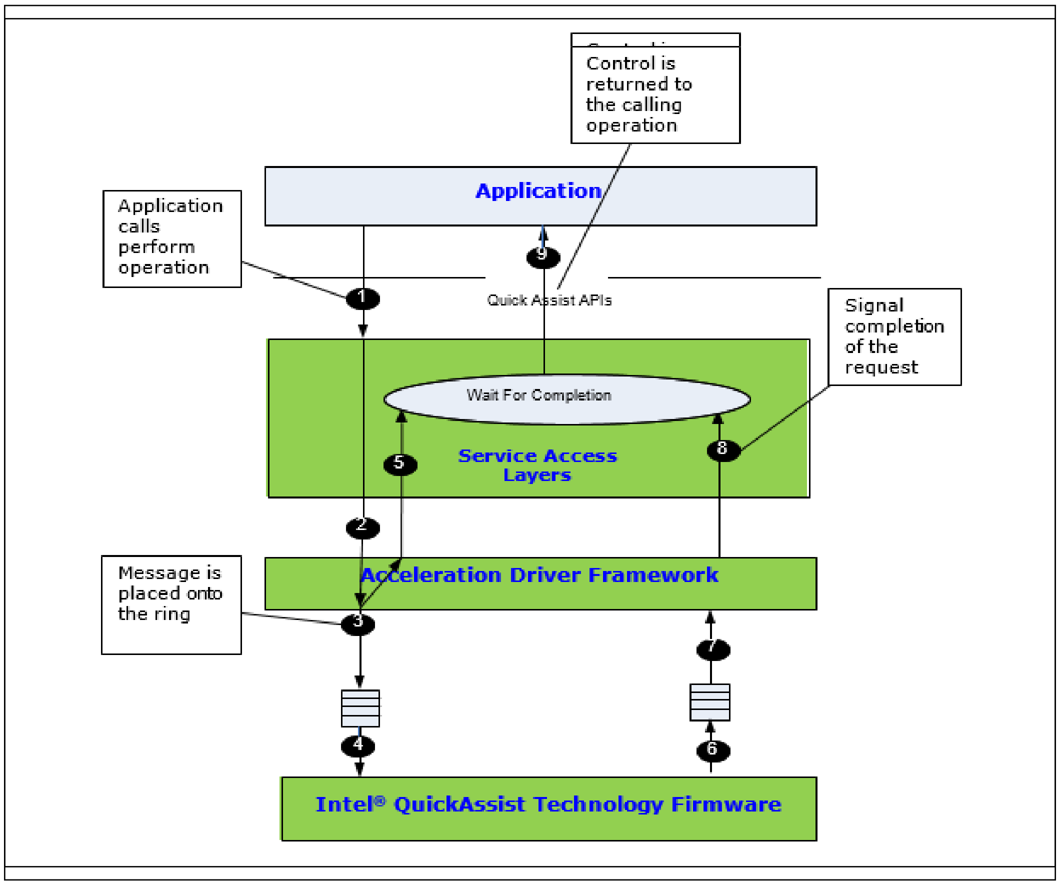
Synchronous operation is specified by supplying a NULL function pointer in the callback parameter of the perform API, as shown in Figure 4. In this case, the function does not return until the operation is complete. The calling thread may spend on a semaphore or other synchronization primitive after sending the request to the execution engine.

通过在 perform API 的回调参数中提供一个空函数指针来指定同步操作，如 Figure 4

Upon the completion of the operation, the synchronization primitive unblocks, and

execution resumes. Synchronous mode is therefore blocking and should not be used when invoking the function from a context in which sleeping is not allowed (for example, an interrupt context on Linux*).
操作完成后，同步原语解除阻塞，执行重新开始。因此，同步模式是阻塞的，当从不允许休眠的上下文(例如，Linux*上的中断上下文)调用函数时，不应使用同步模式。

Figure 4. Synchronous Operation
图 4。同步运行


Control is returned to the calling operation
Application
Application calls perform operation
Quick Assist APIs
Signal completion of the request
Wait For Completion
Service Access Layers
Acceleration Driver Framework
Message is placed onto the ring
Intel® QuickAssist Technology Firmware
1
2
3
4
5
6
7
8
9

2.2.5 Memory Allocation and Ownership

2.2.6 内存分配和所有权

The convention is that all memory needed by an API implementation is allocated outside of that implementation. In other words, the APIs are defined such that the memory needed to execute operations is supplied by a client or platform control entity rather than having memory allocated internally.

惯例是，API 实现所需的所有内存都在该实现之外分配。换句话说，API 被定义为使得执行操作所需的存储器由客户端或平台控制实体提供，而不是在内部分配存储器。

Memory used for parameters is owned by the side (caller or callee) that allocated the memory. An owner is responsible for de-allocating the memory when it is no longer needed.
用于参数的内存归分配内存的一方(调用方或被调用方)所有。当不再需要内存时，所有者负责释放内存。

Generally, memory ownership does not change. For example, if a program allocates

memory and then passes a pointer to the memory as a parameter to a function call, the caller retains ownership and is still responsible for the de-allocation of the memory. Default behavior and any function which deviates from this behavior clearly state so in the function definition.
通常，内存所有权不会改变。例如，如果程序分配内存，然后将指向内存的指针作为参数传递给函数调用，则调用方保留所有权，并仍然负责内存的取消分配。默认行为和任何偏离这种行为的函数都在函数定义中明确说明。

For optimal performance, data pointers should be 8-byte aligned. In some cases, this is a requirement, while in most other cases, it is a recommendation for performance. Refer to Table 1 for the service-specific API manual for optimal usage of the particular API.
为了获得最佳性能，数据指针应该8字节对齐。在某些情况下，这是一个要求，而在大多数其他情况下，这是一个性能建议。涉及 Table 1

2.2.7 Data Plane APIs
2.2.8 数据平面 API

The Intel® QAT APIs for symmetric cryptography and for data compression supports both “traditional” (`cpa_cy_sym.h` and `cpa_dc.h`) and “data plane” APIs (`cpa_cy_sym_dp.h` and `cpa_dc_dp.h`)[1]. The data plane APIs are recommended for applications running in a data plane environment where the cost of offload (that is, the cycles consumed by the driver sending requests to the accelerator) needs to be minimized. Several constraints have been placed on these APIs to minimize the cost of offload. If these constraints are too restrictive for a given application, the more general-purpose "traditional" APIs can be used (at an increased cost of offload).

用于对称加密和数据压缩的英特尔 QAT APIs 支持“传统”(cpa_cy_sym.h 和 cpa_dc.h)和“数据平面”API(CPA _ cy _ sym _ DP . h 和 cpa_dc_dp.h)[1]

The data plane APIs can be used if the following constraints are acceptable:
如果以下约束是可接受的，则可以使用数据平面 API:

- There is no support for partial packets or stateful requests.
- 不支持部分数据包或有状态请求。
- Thread safety is not supported. Each software thread should have access to its unique instance (`CpaInstanceHandle`).
- 不支持线程安全。每个软件线程都应该能够访问其唯一的实例(CpaInstanceHandle)。
- Only asynchronous invocation is supported.
- 仅支持异步调用。
- Polling is used, rather than interrupts, to dispatch callback functions. Callbacks are invoked in the context of a polling thread.
- 使用轮询而不是中断来分派回调函数。回调是在轮询线程的上下文中调用的。
  - The user application is responsible for creating and scheduling this polling thread.
  - 用户应用程序负责创建和调度这个轮询线程。
    Polling functions are not defined by the Intel® QAT API. Implementations provide their polling functions.
    英特尔 QAT API 没有定义轮询功能。实现提供了它们的轮询功能。
    Refer to Table 1 for Implementation Specific Documentation containing further information on polling functions.
    涉及 Table 1
- Buffers and buffer lists are passed using physical addresses to avoid virtual-to-physical-address translation costs.
- 使用物理地址传递缓冲区和缓冲区列表，以避免虚拟到物理地址的转换成本。
- Alignment restrictions may be placed on the operation data (that is, `CpaCySymDpOpData` and `CpaDcDpOpData`) and buffer list (that is, `CpaPhysBufferList`) structures passed to the data plane APIs. For example, the operation data may need to be at least 8-byte aligned, contiguous, resident, DMA-accessible memory. Refer to Table 1 for Implementation Specific Documentation for more details.
- 可以对传递给数据平面 API 的操作数据(即 CpaCySymDpOpData 和 CpaDcDpOpData)和缓冲区列表(即 CpaPhysBufferList)结构进行对齐限制。例如，操作数据可能需要至少 8 字节对齐、连续、驻留、DMA 可访问的存储器。涉及 Table 1

- For CCM and GCM modes of the AES, when performing decryption and verification, if the verification fails, then the message buffer is not zeroed.
- 对于 AES 的 CCM 和 GCM 模式，当执行解密和验证时，如果验证失败，则消息缓冲区不会清零。

The data plane APIs distinguish between enqueuing a request and submitting that request to the accelerator to be performed. This allows the cost of submitting a request (which can be expensive, in terms of cycles, for some hardware-based implementations) to be amortized over all enqueued requests on that instance (`CpaInstanceHandle`).

数据平面 API 区分将请求入队和将该请求提交给加速器来执行。这使得提交请求的成本(对于一些基于硬件的实现来说，在周期方面可能很昂贵)可以分摊到该实例上所有排队的请求中(CpaInstanceHandle)。

- To enqueue one request and to optionally submit all previously enqueued requests, the function `cpaCySymDpEnqueueOp` (or `cpaDcDpEnqueueOp` for data compression service) can be used.
- 要将一个请求入队并选择性地提交所有先前入队的请求，可以使用函数 cpaCySymDpEnqueueOp(或用于数据压缩服务的 cpaDcDpEnqueueOp)。
- To enqueue multiple requests and to optionally submit all previously enqueued requests, the function `cpaCySymDpEnqueueOpBatch` (or `cpaDcDpEnqueueOpBatch` for data compression service) can be used.
- 要将多个请求入队并有选择地提交所有先前入队的请求，可以使用函数 cpaCySymDpEnqueueOpBatch(或用于数据压缩服务的 cpaDcDpEnqueueOpBatch)。

---

[1] There is no "data plane" support for asymmetric cryptography services.

1 不存在对非对称加密服务的"数据平面"支持。

- Use the function `cpaCySymDpPerformOpNow` (or `cpaDcDpPerformOpNow` for data compression service) that can be used to submit all previously enqueued requests.
- 使用函数 cpaCySymDpPerformOpNow(或用于数据压缩服务的 cpaDcDpPerformOpNow),该函数可用于提交所有先前排队的请求。
- Different implementations of this API may have different performance trade-offs. Refer to Table 1 for documentation for implementation details.
- 这个 API 的不同实现可能有不同的性能权衡。涉及 Table 1

§

5 Intel® QuickAssist Technology Cryptographic API

6 英特尔快速辅助技术加密 API

This chapter describes the sample code for the Intel® QuickAssist Technology Cryptographic API, beginning with an API overview, and followed by descriptions of various scenarios to illustrate the usage of the API.

本章描述了英特尔快速辅助技术加密 API 的示例代码，首先是 API 概述，然后是说明 API 用法的各种场景描述。

6.1 Overview

6.2 概观

The Intel® QuickAssist Technology Cryptographic API can be categorized into the following broad areas:

英特尔快速辅助技术加密 API 可分为以下几大类:

- **Common**: This is defined by the file `cpa_cy_common.h`. This includes the functionality for the initialization and shutdown of the service.
- Common:这是由 cpa_cy_common.h 文件定义的，包括服务的初始化和关闭功能。
- **Instance Management**: The file `cpa_cy_im.h` defines the functions for managing instances. A given implementation of the API can present multiple instances of the cryptographic service, each representing a logical or virtual "device". Request order is guaranteed within a given instance of the service.
- 实例管理:文件 cpa_cy_im.h 定义了管理实例的函数。API 的给定实现可以呈现加密服务的多个实例，每个实例代表一个逻辑或虚拟“设备”。在给定的服务实例中保证请求顺序。
- **Symmetric**: The following files constitute the symmetric API:
- 对称:以下文件构成了对称 API:
  - The `cpa_cy_sym.h` file contains the symmetric API, used for ciphers, hashing/message digests, "algorithm chaining" (combining cipher and hash into a single call), and authenticated ciphers.
  - cpa_cy_sym.h 文件包含对称 API，用于密码、哈希/消息摘要、“算法链接”(将密码和哈希合并到一个调用中)和认证密码。
  - The `cpa_sy_sym_dp.h` file also contains the symmetric API, used for ciphers, hashing/message digest, "algorithm chaining" (combining cipher and hash into a single call) and authenticated ciphers. This API is recommended for data plane applications, in which the cost of offload (i.e. the cycles consumed by the API in sending requests to the hardware and processing the responses) needs to be minimized. Several constraints need to be acceptable to the Application. To use this API, these are listed in Section 2.2.4, Data Plane APIs.
  - cpa_sy_sym_dp.h 文件还包含对称 API，用于密码、哈希/消息摘要、“算法链接”(将密码和哈希组合成一个调用)和认证密码。建议将此 API 用于数据平面应用，在这些应用中，卸载成本(即 API 向硬件发送请求和处理响应所消耗的周期)需要最小化。应用程序需要接受几

个约束。要使用此 API，请参见第节 2.2.4Data Plane APIs

- The cpa_cy_key.h file contains the API for key generation for Secure Sockets Layer (SSL) and Transport Layer Security (TLS).
- cpa_cy_key.h 文件包含用于安全套接字层(SSL)和传输层安全性(TLS)的密钥生成的 API。

- **Asymmetric**: The following files constitute the asymmetric API:
- 不对称:以下文件构成了不对称 API:
  - The cpa_cy_rsa.h file defines the API for RSA.
  - cpa_cy_rsa.h 文件定义了 rsa 的 API。
  - The cpa_cy_dsa.h file defines the API for Digital Signature Algorithm (DSA).
  - cpa_cy_dsa.h 文件定义了数字签名算法(dsa)的 API。
  - The cpa_cy_dh.h file defines the API for Diffie-Hellman.
  - cpa_cy_dh.h 文件定义了 Diffie-Hellman 的 API。
  - The cpa_cy_ec.h file defines the API for "base" elliptic curve cryptography.
  - cpa_cy_ec.h 文件定义了“基本”椭圆曲线加密的 API。
  - The cpa_cy_ecdsa.h file defines the API for Elliptic Diffie (EC) DSA.
  - cpa_cy_ecdsa.h 文件定义了椭圆 Diffie (EC) DSA 的 API。
  - The cpa_cy_ecdh.h file defines the API for Elliptic Diffie-Hellman (ECDH).
  - cpa_cy_ecdh.h 文件定义了椭圆 Diffie-Hellman (ECDH)的 API。
  - The cpa_cy_prime.h file defines the API for prime number testing.
  - cpa_cy_prime.h 文件定义了素数测试的 API。
  - The cpa_cy_ln.h file defines the API for a large number of math operations, such as modular exponentiation, etc.
  - cpa_cy_ln.h 文件定义了大量数学运算的 API，比如模幂运算等。
  - The cpa_cy_ecsm2.h file defines the API for SM2 algorithm which is based on Elliptic Curves Cryptography (ECC)
  - cpa_cy_ecsm2.h 文件定义了基于椭圆曲线加密(ECC)的 sm2 算法的 API

- **Random Bit Generation (RBG)**: The following files constitute the RBG API and have been deprecated because random bit generation can be handled in the CPU:
- 随机位生成(RBG):以下文件构成了 RBG API，但已经被弃用，因为随机位生成可以在 CPU 中处理:
  - The cpa_cy_drbg.h file defines the API for deterministic random bit generation.
  - cpa_cy_drbg.h 文件定义了用于确定性随机位生成的 API。

- The `cpa_cy_nrbg.h` file defines the API for a non-deterministic random bit generation.
- cpa_cy_nrbg.h 文件定义了用于非确定性随机位生成的 API。

The Cryptographic API uses the base API, which defines base data types used across all services of the Intel® QAT API.
加密 API 使用基本 API，该 API 定义了英特尔 QAT API 所有服务中使用的基本数据类型。

3.1.1 Sessions
3.1.2 会议

The symmetric API is the only API with the concept of sessions. The meaning of a session within the symmetric API is defined below.

对称 API 是唯一具有会话概念的 API。对称 API 中会话的含义定义如下。

3.1.3 Priority
3.1.4 优先

The Cryptographic symmetric API has support for priorities. Priority can be specified on a per-session basis. Two levels of priority are supported: high priority and normal priority. Implementations may use a strict priority order or a weighted round robin-based priority scheme.

加密对称 API 支持优先级。可以在每个会话的基础上指定优先级。支持两种优先级:高优先级和普通优先级。实现可以使用严格的优先级顺序或基于加权循环的优先级方案。

6.3 Using the Symmetric Cryptography API
6.4 使用对称加密 API

This section contains examples of how to use the symmetric API. It describes general concepts and how to use the symmetric API to perform various types of cipher and hash.

本节包含如何使用对称 API 的示例。它描述了一般概念以及如何使用对称 API 来执行各种类型的加密和散列。

Note: Examples are simplified and demonstrate how to use the APIs and build the structures required for various use cases.
注意:示例是简化的，演示了如何使用 API 和构建各种用例所需的结构。

These examples may not demonstrate the optimal way to use the API to get maximum performance for a particular implementation. Refer to Table 1 for Implementation Specific Documentation and performance sample code for a guide on how to use the API for best performance.

这些例子可能没有展示使用 API 来获得特定实现的最佳性能的最佳方式。涉及 Table 1

Note: All of the symmetric examples follow the same basic steps:
注意:所有对称示例都遵循相同的基本步骤:

- Define a callback function (if the API is to be invoked asynchronously)
- 定义一个回调函数(如果 API 将被异步调用)
- Discover and start up the cryptographic service instance
- 发现并启动加密服务实例
- Create and initialize a session
- 创建并初始化会话
- Invoke multiple symmetric operations (cipher and/or hash) on the session
- 在会话上调用多个对称操作(密码和/或哈希)
- Tear down the session
- 拆除会话
- Stop the Cryptographic service instance
- 停止加密服务实例

3.2.1 General Concepts
3.2.2 一般概念

This section describes the following concepts:

本节描述以下概念:

- Session
- 会议
- Place and Out-of-Place Support
- 位置和非位置支持

- Partial Support
- 部分支持

3.2.1.1 Session
3.2.1.2 会议

In case of the symmetric API, a session is a handle that describes the cryptographic parameters to be applied to several buffers. This might be the buffers within a single file or all the packets associated with a particular Internet Protocol Security (IPSec) tunnel or security association. The data within a session handle includes the following:

在对称 API 的情况下，会话是描述要应用于几个缓冲区的加密参数的句柄。这可能是单个文件中的缓冲区，也可能是与特定互联网协议安全(IPSec)隧道或安全关联相关联的所有数据包。会话句柄中的数据包括以下内容:

- The operation (cipher, hash, or both, and if both, the order in which the algorithms should be applied).
- 操作(密码、哈希或两者都有，如果两者都有，算法的应用顺序)。
- The cipher setup data, including the cipher algorithm and mode, the key and its length, and the direction (encrypt or decrypt).
- 密码设置数据，包括密码算法和模式、密钥及其长度以及方向(加密或解密)。
- The hash setup data, including the hash algorithm, mode (plain, nested or authenticated), and digest result length (to allow for truncation).
- 哈希设置数据，包括哈希算法、模式(普通、嵌套或认证)和摘要结果长度(允许截断)。
  - The authenticated mode can refer to Hashed Message Authenticate Code (HMAC), which requires that the key and its length are also specified. It is also used for Galois Counter Mode (GCM), and Counter mode with Cipher-block Chaining Message authentication code (CCM) authenticated encryption, in which case the Additional Authenticated Data (AAD) length is also specified.
  - 认证模式可以参考散列消息认证代码(HMAC),这要求还指定密钥及其长度。它还用于伽罗瓦计数器模式(GCM)和带密码块链接消息认证码(CCM)认证加密的计数器模式，在这种情况下，还规定了附加认证数据(AAD)长度。
  - For nested mode, the inner and outer prefix data and length are specified, as well as the outer hash algorithm.
  - 对于嵌套模式，指定了内部和外部前缀数据和长度，以及外部哈希算法。

3.2.1.3 In-Place and Out-of-Place Support
3.2.1.4 就地和非就地支持

An In-Place operation means that the destination buffer is the same as the source buffer. An Out-of-Place operation means that the destination buffer is different from the source buffer.

就地操作意味着目标缓冲区与源缓冲区相同。错位操作意味着目标缓冲区不同于源缓冲区。

3.2.1.5 Partial Support
3.2.1.6 部分支持

Most of the examples in this chapter operate on full packets, as indicated by the `packetType` of `CPA_CY_SYM_PACKET_TYPE_FULL`. The API also supports operating in partial mode, where, for example, state (e.g., cipher state) needs to be carried forward from one packet/record to the next. In Section 3.2.4, Hash a File, there is an example of hashing a file that uses the partial API.

本章中的大多数示例都是对完整的数据包进行操作的，如CPA_CY_SYM_PACKET_TYPE_FULL的数据包类型所示。API还支持在部分模式下操作，其中，例如，状态(例如，密码状态)需要从一个分组/记录结转到下一个。在截面中3.2.4Hash a File

Note:

注意:

1. The size of the data to be hashed or ciphered must be a multiple of the block size of the algorithm for all partial packets.
2. 要被散列或加密的数据的大小必须是所有部分分组的算法的块大小的倍数。
3. For hash/authentication, the digest verify flag only applies to the last partial packet.
4. 对于哈希/认证，摘要验证标志仅适用于最后一个部分数据包。
5. For algorithm chaining, only the cipher state is maintained between calls. The hash state is not maintained between calls; instead, the hash digest is generated/verified for each call. The size of the data to be ciphered must be a multiple of the block size of the algorithm for all partial packets. The size of the data to be hashed does not have this restriction. If both the cipher state and the hash state need to be maintained between calls, then algorithm chaining cannot be used.
6. 对于算法链，在调用之间只维护密码状态。在调用之间不维护哈希状态；相反，为每个调用生成/验证散列摘要。要加密的数据的大小必须是所有部分分组的算法的块大小的倍数。要散列的数据的大小没有这种限制。如果在调用之间需要维护密码状态和散列状态，则不能使用算法链。

3.2.3 Cipher
3.2.4 零

This example demonstrates the usage of the symmetric API, specifically using this API to perform a cipher operation. It encrypts some sample text using the AES-256 algorithm in Cipher Block Chaining (CBC) mode.

这个例子演示了对称 API 的用法，特别是使用这个 API 来执行加密操作。它在密码块链接(CBC)模式下使用 AES-256 算法加密一些样本文本。

These samples are located in:
这些样本位于:

`quickassist/lookaside/access_layer/src/sample_code/functional/sym/`

quickassist/lookaside/access_layer/src/sample_code/functional/sym/cipher_ sample

The following subsections describe the main functions in this file.

以下小节描述了该文件中的主要函数。

3.2.2.1 symCallback
3.2.2.2 符号回调

A callback function must be supplied to use the API in asynchronous mode, and this function is called back (that is, invoked by the implementation of the API) when the asynchronous operation has completed. The context in which it is invoked depends on the implementation. For example, it could be invoked in the context of a Linux* interrupt handler's bottom half or in the context of a user created polling thread. The context in which this function is invoked places restrictions on what processing can be done in the callback function. On the API, it states that this function should not sleep (since it may be called in a context that does not permit sleeping, for example, a Linux* bottom half).

要在异步模式下使用 API，必须提供一个回调函数，当异步操作完成时，该函数被回调(即由 API 的实现调用)。调用它的上下文取决于实现。例如，它可以在 Linux*中断处理程序下半部分的上下文中调用，或者在用户创建的轮询线程的上下文中调用。调用该函数的上下文限制了回调函数中可以完成的处理。在 API 上，它声明该函数不应休眠(因为它可能在不允许休眠的上下文中被调用，例如 Linux*下半部分)。

This function can perform whatever processing is appropriate for the Application. For example, it may free memory, continue the processing of a decrypted packet, etc. In this example, the function only sets the complete variable to indicate it has been called, as illustrated below.
这个函数可以执行任何适合应用程序的处理。例如，它可以释放存储器，继续处理解密的分组等。在本例中，该函数仅设置 complete 变量来指示它已被调用，如下所示。

Listing 3. Callback Function
清单 3。回调函数

```
static void symCallback(void *pCallbackTag,
                        CpaStatus status,
                        const CpaCySymOp
                        operationType, void *pOpData,
                        CpaBufferList
                        *pDstBuffer, CpaBoolean
                        verifyResult)
{
    PRINT_DBG("Callback called with status = %d.\n", status);

    if (NULL != pCallbackTag) {
```

static void symCallback(void *pCallbackTag,
CpaStatus status,
const CpaCySymOp operationType,
void *pOpData,
CpaBufferList *pDstBuffer,
CpaBoolean verifyResult)
{
PRINT_DBG("Callback called with status = %d.\n", status);
if (NULL != pCallbackTag) {

3.2.2.3 cipherSample
3.2.2.4 密码样本

This is the main entry point for the sample cipher code. It demonstrates the sequence of calls to be made to the API to create a session, perform one or more cipher operations, and then tear down the session. The following is performed:

这是示例密码的主要入口点。它演示了调用 API 来创建会话、执行一个或多个加密操作，然后拆除会话的顺序。将执行以下操作：

- Call the instance discovery utility function - `sampleCyGetInstance` - which is a simplified version of instance discovery, in which exactly one instance of a crypto service is discovered. It does this by querying the API for all instances, and returning the first instance, as illustrated in Listing 4.
- 调用实例发现实用程序函数 sampleCyGetInstance，这是实例发现的简化版本，其中只发现一个加密服务实例。它通过在 API 中查询所有实例并返回回第一个实例来实现这一点，如 Listing 4
- This step is described in Section 2.2.1, Instance Discovery, but is repeated here for convenience.
- 第节描述了这一步骤 2.2.1Instance Discovery

Listing 4. Getting an Instance
清单 4。获取实例

```
#ifdef DO_CRYPTO
void sampleCyGetInstance(CpaInstanceHandle* pCyInstHandle)
{
    CpaInstanceHandle
    cyInstHandles[MAX_INSTANCES]; Cpa16U
    numInstances = 0;
    CpaStatus status = CPA_STATUS_SUCCESS;

    *pCyInstHandle = NULL;
    status = cpaCyGetNumInstances(&numInstances);
    if ((status == CPA_STATUS_SUCCESS) && (numInstances > 0))
    {
        status = cpaCyGetInstances(MAX_INSTANCES,
        cyInstHandles); if (status == CPA_STATUS_SUCCESS)
        {
            *pCyInstHandle = cyInstHandles[0];
        }
    }
    if (0 == numInstances)
    {
        PRINT_ERR("No instances found for 'SSL'\n");
        PRINT_ERR("Please check your section names in the config
file.\n");
        PRINT_ERR("Also make sure to use config file version 2.\n");
    }
}
#endif
```

```
#ifdef DO_CRYPTO
void sampleCyGetInstance(CpaInstanceHandle* pCyInstHandle)
{
    CpaInstanceHandle cyInstHandles[MAX_INSTANCES]; Cpa16U
    numInstances = 0;
    CpaStatus status = CPA_STATUS_SUCCESS;

    *pCyInstHandle = NULL;
    status = cpaCyGetNumInstances(&numInstances);
    if ((status == CPA_STATUS_SUCCESS) && (numInstances > 0))
    {
        status = cpaCyGetInstances(MAX_INSTANCES, cyInstHandles); if
        (status == CPA_STATUS_SUCCESS)
        {
            *pCyInstHandle = cyInstHandles[0];
        }
    }
    if (0 == numInstances)
    {
        PRINT_ERR("No instances found for 'SSL'\n");
        PRINT_ERR("Please check your section names in the config file.\n");
        PRINT_ERR("Also make sure to use config file version 2.\n");
    }
}
#endif
```

Set the address translation function for the instance. This function will be used by the API to convert virtual addresses to physical addresses.

为实例设置地址转换功能。API 将使用该函数将虚拟地址转换为物理地址。

Listing 5. Set Address Translation Function
清单 5。设置地址转换功能

```
status = cpaCySetAddressTranslation(cyInstHandle,
```

```
status = cpaCySetAddressTranslation(cyInstHandle,
```

Start the crypto service running as shown below.

启动如下所示运行的加密服务。

Listing 6. Start up
清单 6。开始

```
status = cpaCyStartInstance(cyInstHandle);
```

```
status = cpaCyStartInstance(cyInstHandle);
```

The next step is to create and initialize a session. First, populate the fields of the session initialization operational data structure.

下一步是创建和初始化会话。首先，填充会话初始化操作数据结构的字段。

Note: The size required to store a session is implementation-dependent, so you must query the API first to determine how much memory to allocate, and then allocate that memory.
注意:存储一个会话所需的大小取决于实现，因此您必须首先查询 API 以确定要分配多少内存，然后分配该内存。

One of two available queries can be used:
可以使用两个可用查询之一:

- `cpaCySymSessionCtxGetSize(const CpaInstanceHandle instanceHandle_in, const CpaCySymSessionSetupData *pSessionSetupData, Cpa32U`
- cpacysymsessiontxgetsize(const CpaInstanceHandle instance handle _ in, const CpaCySymSessionSetupData * pSessionSetupData, Cpa32U
  `*pSessionCtxSizeInBytes)`
  * pSessionCtxSizeInBytes)
- This will always return the maximum session context size (i.e., the full size of the session including padding and other session state information) (see Listing 7 below).
- 这将总是返回最大的会话上下文大小(即会话的完整大小，包括填充和其他会话状态信息)(参见下面的清单 7)。
- `cpaCySymSessionCtxGetDynamicSize(const CpaInstanceHandle instanceHandle_in, const CpaCySymSessionSetupData *pSessionSetupData, Cpa32U *pSessionCtxSizeInBytes)`
- cpacysymsessiontxgetdynamicsize(const CpaInstanceHandle instance handle _ in, const CpaCySymSessionSetupData * pSessionSetupData, Cpa32U * pSessionCtxSizeInBytes)
- This query can be used instead to return a reduced memory size, based on whether the use case meets certain session setup criteria (see Listing 7 below).
- 根据用例是否满足特定的会话设置标准，此查询可用于返回减少的内存大小(参见 Listing 7
- This query will return one of three values for `pSessionCtxSizeInBytes` as follows:
- 该查询将返回 pSessionCtxSizeInBytes 的三个值之一，如下所示:
  - If partial packets are not being used and the Symmetric operation is Auth-Encrypt (i.e., the cipher and hash algorithms are either CCM or GCM), the size returned will be approximately half of the standard size.
  - 如果没有使用部分数据包，并且对称操作是 Auth- Encrypt(即，密码和哈希算法是 CCM 或 GCM)，则返回的大小大约是标准大小的一半。
  - If partial packets are not being used and the cipher algorithm is not `ARC4`, `Snow3g_UEA2`, `AES_CCM` or `AES_GCM`, and the hash algorithm is not `Snow3G_UIA2`, `AES_CCM` or `AES_GCM`, and Hash Mode is not Auth, the size returned will be between half and one third of the standard size.
  - 如果未使用部分数据包，并且密码算法不是 ARC4、Snow3g_UEA2、AES_CCM 或 AES_GCM，哈希算法不是 Snow3G_UIA2、AES_CCM 或 AES_GCM，并且哈希模式不是 Auth，则返回的大小将介于标准大小的一半和三分之一之间。
  - In all other cases, the standard size is returned.
  - 在所有其他情况下，返回标准大小。

Note: The following parameter exists in the `CpaCySymSessionSetupData structure`:

`CpaBoolean partialsNotRequired`

注意:以下参数存在于CpaCySymSessionSetupData结构中:CpaBoolean partialsNotRequired

This flag indicates if partial packet processing is required for the session. If partial packets are not being used and the preference is to use one of the reduced session memory sizes, set this flag to `CPA_TRUE` before calling the `cpaCySymSessionCtxGetDynamicSize()` function.

此标志指示会话是否需要部分数据包处理。如果没有使用部分数据包，并且首选使用一个减小的会话内存大小，请在调用cpaCySymSessionCtxGetDynamicSize()函数之前将此标志设置为CPA_TRUE。

Note: The equivalent reduced memory context query for Data Plane API (Refer to Section 3.2.10, Chained Cipher and Hash Using the Symmetric Data Plane API is:

*注意:针对数据平面API的等效精简内存上下文查询(请参考部分*3.2.10Chained Cipher and Hash Using the Symmetric Data Plane API

```
cpaCySymDpSessionCtxGetDynamicSize(const CpaInstanceHandle
instanceHandle_in, const CpaCySymSessionSetupData *pSessionSetupData,
Cpa32U *pSessionCtxSizeInBytes)
```

cpacysymdpsessiontxgetdynamicsize(const CpaInstanceHandle instance handle _ in, const CpaCySymSessionSetupData * pSessionSetupData, Cpa32U * pSessionCtxSizeInBytes)

Listing 7. Create and Initialize Cipher Session

清单7。创建并初始化密码会话

```
/* Populate the session setup structure for the operation required */
/* Populate the session setup structure for the operation required */
sessionSetupData.sessionPriority = CPA_CY_PRIORITY_NORMAL;
sessionSetupData.sessionPriority = CPA_CY_PRIORITY_NORMAL;
sessionSetupData.symOperation = CPA_CY_SYM_OP_CIPHER;
sessionSetupData.symOperation = CPA_CY_SYM_OP_CIPHER;
sessionSetupData.cipherSetupData.cipherAlgorithm =
sessionSetupData.cipherSetupData.cipherAlgorithm = CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection = CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT

/* Determine size of session context to allocate */ PRINT_DBG("cpaCySymSessionCtxGetSi
status =
cpaCySymSessionCtxGetSize(cyInstHandle, &sessionSetupData, &sessionCtxSize);
}

if (CPA_STATUS_SUCCESS == status) {
/* Allocate session context */
status = PHYS_CONTIG_ALLOC(&sessionCtx, sessionCtxSize);
}









/* Initialize the Cipher session */ if (CPA_STATUS_SUCCESS == status) {
PRINT_DBG("cpaCySymInitSession\n");
status = cpaCySymInitSession(cyInstHandle,
symCallback,
*/
                                                    /* callback function

                                 &sessionSetupData, /* session setup data
*/
                                 sessionCtx);       /* output of the
function*/
}
```

```
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection = CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;

/* Determine size of session context to allocate */ PRINT_DBG("cpaCySymSessionCtxGetSize\n");
status =
cpaCySymSessionCtxGetSize(cyInstHandle, &sessionSetupData, &sessionCtxSize);
}

if (CPA_STATUS_SUCCESS == status) {
/* Allocate session context */
status = PHYS_CONTIG_ALLOC(&sessionCtx, sessionCtxSize);
}









/* Initialize the Cipher session */ if (CPA_STATUS_SUCCESS == status) {
PRINT_DBG("cpaCySymInitSession\n");
status = cpaCySymInitSession(cyInstHandle,
symCallback,
*/
                                                            /* callback function

                                      &sessionSetupData, /* session setup data
*/
                                      sessionCtx);          /* output of the
function*/
}
```

- Call the function cipherPerformOp, which actually performs the cipher operation. This in turn performs the following steps:
- 调用 cipherPerformOp 函数，该函数实际执行加密操作。这又会执行以下步骤:

Memory Allocation: Different implementations of the API require different amounts of space to store metadata associated with buffer lists. Query the API to find out how much space the current implementation needs, and then allocate space for the buffer metadata, the buffer list, and for the buffer itself. You must also allocate memory for the initialization vector.

内存分配:API 的不同实现需要不同的空间来存储与缓冲区列表相关的元数据。查询 API 以找出当前实现需要多少空间，然后为缓冲区元数据、缓冲区列表和缓冲区本身分配空间。您还必须为初始化向量分配内存。

Listing 8. Memory Allocation
清单 8。存储器分配

```
status = cpaCyBufferListGetMetaSize(cyInstHandle, numBuffers,
&bufferMetaSize);
if (CPA_STATUS_SUCCESS == status) {
    status = PHYS_CONTIG_ALLOC(&pBufferMeta, bufferMetaSize);
}
if (CPA_STATUS_SUCCESS == status) {
    status = OS_MALLOC(&pBufferList, bufferListMemSize);
}
```

```
status = cpaCyBufferListGetMetaSize(cyInstHandle, numBuffers, &bufferMetaSize);
if (CPA_STATUS_SUCCESS == status) {
    status = PHYS_CONTIG_ALLOC(&pBufferMeta, bufferMetaSize);
}
if (CPA_STATUS_SUCCESS == status) {
    status = OS_MALLOC(&pBufferList, bufferListMemSize);
}
if (CPA_STATUS_SUCCESS == status) {
    status = PHYS_CONTIG_ALLOC(&pSrcBuffer, bufferSize);
}
if (CPA_STATUS_SUCCESS == status) {
    status = PHYS_CONTIG_ALLOC(&pIvBuffer, sizeof(sampleCipherIv));
```

```
    status = PHYS_CONTIG_ALLOC(&pSrcBuffer, bufferSize);
}
if (CPA_STATUS_SUCCESS == status) {
    status = PHYS_CONTIG_ALLOC(&pIvBuffer, sizeof(sampleCipherIv));
```

- **Memory Allocation**: The memory for the source buffer and initialization vector is populated with the required data.
- 内存分配:用于源缓冲区和初始化向量的内存由所需的数据填充。
- **Set Up Operational Data**: Populate the structure containing the operational data that is needed to run the algorithm as shown below.
- 设置操作数据:填充包含运行算法所需的操作数据的结构，如下所示。

Listing 9. Set Up Cipher Operational Data
清单 9。设置密码操作数据

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType =
CPA_CY_SYM_PACKET_TYPE_FULL; pOpData->pIv =
pIvBuffer;
pOpData->ivLenInBytes =
```

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
pOpData->pIv = pIvBuffer;
pOpData->ivLenInBytes = sizeof(sampleCipherIv);
pOpData->cryptoStartSrcOffsetInBytes = 0;
```

- **Perform Operation**: Initialize the completion variable, which is used by the callback function to indicate that the operation is complete, then perform the operation.
- 执行操作:初始化完成变量，回调函数使用该变量来指示操作已完成，然后执行操作。

Listing 10. Perform Cipher Operation
清单 10。执行密码运算

```
COMPLETION_INIT(&complete);

status =
    cpaCySymPerformOp( cy
    instHandle,
    (void *)&complete, /* data sent as is to the callback
    function*/ pOpData,         /* operational data struct */
    pBufferList,         /* source buffer list */
```

```
COMPLETION_INIT(&complete);
status = cpaCySymPerformOp(
    cyInstHandle,
    (void *)&complete, /* data sent as is to the callback function*/ pOpData,
                       /* operational data struct */
    pBufferList,       /* source buffer list */
    pBufferList,       /* same src & dst for an in-place operation*/ NULL);
```

- **Wait for Completion**: Because the asynchronous API is used in this example, the callback function must be handled. This example uses a macro that can be defined differently for different operating systems. In a typical real-world application, the calling thread would not block, and the callback would essentially re-inject the (decrypted, decapsulated) packet into the stack.
- 等待完成:因为本例中使用了异步API，所以必须处理回调函数。此示例使用了一个宏，该宏可以针对不同的操作系统进行不同的定义。在一个典型的实际应用程序中，调用线程不会阻塞，回调会将(解密、解封装的)数据包重新注入堆栈。

Listing 11. Wait for Completion
清单 11。等待完成

```
if (!COMPLETION_WAIT(&complete, TIMEOUT_MS)) {
    PRINT_ERR("timeout or interruption in cpaCySymPerformOp\
    n"); status = CPA_STATUS_FAIL;
```

```
if (!COMPLETION_WAIT(&complete, TIMEOUT_MS)) { PRINT_ERR("timeout or
    interruption in cpaCySymPerformOp\n"); status = CPA_STATUS_FAIL;
}
```

In a normal usage scenario, the session would be reused multiple times to encrypt multiple buffers or packets. In this example, however, the session is torn down.

在正常使用情况下，会话将被多次重用，以加密多个缓冲区或数据包。但是，在本例中，会话被拆除。

Listing 12. Wait for Outstanding Requests

清单 12。等待未完成的请求

```
symSessionWaitForInflightReq(sessionCtx)
```

```
symSessionWaitForInflightReq(session CTX)
```

Since cryptographic API v2.2 before removing the symmetric session context it is recommended to wait for the completion of any outstanding request using `cpaCySymSessionInUse`.

由于加密 API v2.2，在删除对称会话上下文之前，建议使用 cpaCySymSessionInUse 等待任何未完成的请求完成。

It is executed in the `symSessionWaitForInflightReq` call which polls for the in-flight requests.

它在 symSessionWaitForInflightReq 调用中执行，该调用轮询正在进行的请求。

Listing 13. Remove Cipher Session

清单 13。删除密码会话

```
sessionStatus = cpaCySymRemoveSession(cyInstHandle, sessionCtx);
```

```
session status = cpaCySymRemoveSession(cyInstHandle, session CTX);
```

- Query statistics at this point, which can be useful for debugging.
- 此时查询统计信息，这对调试很有用。

  Some implementations may also make the statistics available through other mechanisms, such as the `/proc` virtual filesystem.

  一些实现还可能通过其他机制(如/proc 虚拟文件系统)来提供统计信息。
- Finally, clean up by freeing up memory, stopping the instance, etc.
- 最后，通过释放内存、停止实例等方式进行清理。
- Since Cryptographic API v2.2 two new functions have been implemented:
- 自从加密 API v2.2 以来，已经实现了两个新功能:

```
cpaCySymUpdateSession and cpaCySymSessionInUse
```

```
cpaCySymUpdateSession 和 cpaCySymSessionInUse
```

- The function `cpaCySymUpdateSession` can be used to update certain parameters of a session like the cipher key, the cipher direction, and the authentication key. `cpaCySymSessionInUse`, indicates whether there are outstanding requests on a given session.
- 函数 cpaCySymUpdateSession 可用于更新会话的某些参数，如加密密钥、加密方向和身份验证密钥。cpaCySymSessionInUse，指示给定会话中是否有未完成的请求。

  As a result of the implementation of this feature, the behavior of `cpaCySymRemoveSession` has been changed. `cpaCySymRemoveSession` will fail if there are outstanding request for the session that the user is trying to remove.

  由于实现了此功能，cpaCySymRemoveSession 的行为已被更改。如果用户试图删除的会话有未完成的请求，则 cpaCySymRemoveSession 将失败。

As a result, it is recommended to wait for the completion of any outstanding request, using `cpaCySymSessionInUse,` before removing a session.

因此，建议在删除会话之前，使用 cpaCySymSessionInUse 等待任何未完成的请求完成。

3.2.5 Hash
3.2.6 混杂

This example demonstrates the usage of the symmetric API, specifically using this API to perform a hash operation. It performs an SHA-256 hash operation on some sample data.

这个例子演示了对称 API 的用法，特别是使用这个 API 来执行散列操作。它对一些样本数据执行 SHA-256 哈希运算。

These samples are located in `/sym/hash_sample`
这些示例位于/sym/hash_sample 中

The example is very similar to the cipher example, so only the differences are highlighted:

该示例与 cipher 示例非常相似，因此只强调不同之处:

When creating and initializing a session, some of the fields of the session initialization operational data structure are different from the cipher case, as shown below.
创建和初始化会话时，会话初始化操作数据结构的一些字段不同于密码情况，如下所示。

Listing 14. Create and Initialize Hash Session
清单 14。创建并初始化哈希会话

```
/* populate symmetric session data structure
 * for a plain hash operation */
sessionSetupData.sessionPriority =
```

```
/* populate symmetric session data structure
 * for a plain hash operation */ sessionSetupData.sessionPriority =
CPA_CY_PRIORITY_NORMAL; sessionSetupData.symOperation =
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA256;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_PLAIN;
sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
/* Place the digest result in a buffer unrelated to srcBuffer */
sessionSetupData.digestIsAppended = CPA_FALSE;
/* Generate the digest */
```

```
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA256;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_PLAIN;
sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
/* Place the digest result in a buffer unrelated to srcBuffer */
sessionSetupData.digestIsAppended = CPA_FALSE;
/* Generate the digest */
```

When calling the function to perform the hash operation, some of the fields of the operational data structure are again different from the cipher case, as shown below.

当调用用函数来执行散列操作时，操作数据结构的一些字段再次不同于密码情况，如下所示。

Listing 15. Set up Hash Operational Data
清单 15。设置散列操作数据

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType =
CPA_CY_SYM_PACKET_TYPE_FULL; pOpData-
>hashStartSrcOffsetInBytes = 0;
```

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
pOpData->hashStartSrcOffsetInBytes = 0;
pOpData->messageLenToHashInBytes = sizeof(vectorData);
```

3.2.7 Hash a File

3.2.8 散列文件

This example demonstrates the usage of the symmetric API for partial mode, specifically using this API to perform hash operations. It performs a SHA1 hash operation on a file.

这个例子演示了对称 API 在部分模式下的用法，特别是使用这个 API 来执行散列操作。它对文件执行 SHA1 哈希运算。

These samples are located in /sym/hash_file_sample
这些示例位于/sym/hash_file_sample 中

The example is very similar to the cipher example, so only the differences are highlighted:

该示例与 cipher 示例非常相似，因此只强调不同之处:

When creating and initializing a session, some of the fields of the session initialization operational data structure are different from the cipher case, as shown below.
创建和初始化会话时，会话初始化操作数据结构的一些字段不同于密码情况，如下所示。

Listing 16. Hash Session Setup Data
清单 16。哈希会话设置数据

```
/* populate symmetric session data structure
 * for a plain hash operation */
sessionSetupData.sessionPriority =
CPA_CY_PRIORITY_NORMAL; sessionSetupData.symOperation =
CPA_CY_SYM_OP_HASH;
```

- Memory is allocated for the source buffer in a similar way to the cipher case.
- 为源缓冲区分配内存的方式与加密情况类似。
- To perform the operation data is read from the file to the source buffer and the symmetric API is called repeatedly with `packetType` set to `CPA_CY_SYM_PACKET_TYPE_PARTIAL`. When the end of the file is reached the API is
- 为了执行该操作，将数据从文件读取到源缓冲区，并且在 packetType 设置为 CPA_CY_SYM_PACKET_TYPE_PARTIAL 的情况下重复调用对称 API。当到达文件末尾时，API

```
/* populate symmetric session data structure
 * for a plain hash operation */ sessionSetupData.sessionPriority =
CPA_CY_PRIORITY_NORMAL; sessionSetupData.symOperation =
CPA_CY_SYM_OP_HASH;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA1;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_PLAIN;
sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
/* Place the digest result in a buffer unrelated to srcBuffer */
sessionSetupData.digestIsAppended = CPA_FALSE;
/* Generate the digest */
```

called with packetType set to CPA_CY_SYM_PACKET_TYPE_PARTIAL_LAST. The digest is produced only on the last call to the API.
在 packetType 设置为 CPA _ CY _ SYM _ PACKET _ TYPE _ PARTIAL _ LAST 的情况下调用。摘要仅在最后一次调用 API 时生成。

Listing 17. Hashing a File
清单 17。散列文件

```
while (!feof(srcFile)) {
    /* read from file into src buffer */
    pBufferList->pBuffers->dataLenInBytes
    =
        fread(pSrcBuffer, 1, SAMPLE_BUFF_SIZE, srcFile);

    /* If we have reached the end of file set the last partial flag
    */ if (feof(srcFile)) {
        pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_LAST_PARTIAL;
} else {
        pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_PARTIAL;
    }
    pOpData->sessionCtx = sessionCtx;
    pOpData->hashStartSrcOffsetInBytes =
    0;
    pOpData->messageLenToHashInBytes = pBufferList->pBuffers-
>dataLenInBytes;

    pOpData->pDigestResult = pDigestBuffer;

    PRINT_DBG("cpaCySymPerformOp\n");
    /** Perform symmetric operation */
    status = cpaCySymPerformOp(
        cyInstHandle,
        (void *)&complete, /* data sent as is to the callback
        function*/ pOpData, /* operational data struct */
        pBufferList, /* source buffer list */
        pBufferList, /* same src & dst for an in-place operation*/
        NULL);

    if (CPA_STATUS_SUCCESS != status) {
        PRINT_ERR("cpaCySymPerformOp failed. (status = %d)\n", status);
        break;
    }

    if (CPA_STATUS_SUCCESS == status) {
        /** wait until the completion of the
        operation*/ if (!COMPLETION_WAIT((&complete),
        TIMEOUT_MS)) {
            PRINT_ERR("timeout or interruption in cpaCySymPerformOp\
            n"); status = CPA_STATUS_FAIL;
            break;
```

```
while (!feof(srcFile)) {
    /* read from file into src buffer */
    pBufferList->pBuffers->dataLenInBytes =
        fread(pSrcBuffer, 1, SAMPLE_BUFF_SIZE, srcFile);

    /* If we have reached the end of file set the last partial flag */ if
    (feof(srcFile)) {
        pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_LAST_PARTIAL;
} else {
        pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_PARTIAL;
    }
    pOpData->sessionCtx = sessionCtx; pOpData-
    >hashStartSrcOffsetInBytes = 0;
    pOpData->messageLenToHashInBytes = pBufferList->pBuffers-
>dataLenInBytes;
    pOpData->pDigestResult = pDigestBuffer;
    PRINT_DBG("cpaCySymPerformOp\n");
    /** Perform symmetric operation */ status =
    cpaCySymPerformOp(
        cyInstHandle,
        (void *)&complete, /* data sent as is to the callback function*/ pOpData, /*
        operational data struct */
        pBufferList, /* source buffer list */
        pBufferList, /* same src & dst for an in-place operation*/ NULL);
    if (CPA_STATUS_SUCCESS != status) {
        PRINT_ERR("cpaCySymPerformOp failed. (status = %d)\n", status); break;
    }
    if (CPA_STATUS_SUCCESS == status) {
        /** wait until the completion of the operation*/ if (!
        COMPLETION_WAIT((&complete), TIMEOUT_MS)) {
            PRINT_ERR("timeout or interruption in cpaCySymPerformOp\n"); status =
            CPA_STATUS_FAIL;
            break;
        }
    }
}
```

3.2.9 Chained Cipher and Hash
3.2.10 链式密码和散列

This example demonstrates the usage of the symmetric API, specifically using this API to perform a "chained" cipher and hash operation. It encrypts some sample text using the AES-256 algorithm in CBC mode, and then performs an SHA-256 Hashed Message Authenticate Code (HMAC) operation on the ciphertext, writing the Message Authentication Code (MAC) to the buffer immediately after the ciphertext.

这个例子演示了对称 API 的用法，特别是使用这个 API 来执行“链式”加密和散列操作。它在 CBC 模式下使用 AES-256 算法加密一些样本文本，然后对密文执行 SHA-256 哈希消息验证码(HMAC)操作，在密文后立即将消息验证码(MAC)写入缓冲区。

These samples are located in `/sym/alg_chaining_sample`
这些示例位于/sym/alg_chaining_sample 中

The example is very similar to the cipher and hash examples, above, so only the differences are highlighted:

该示例与上面的密码和哈希示例非常相似，因此只突出显示不同之处：

When creating and initializing a session, some of the fields of the session initialization operational data structure are different, as shown below.
创建和初始化会话时，会话初始化操作数据结构的一些字段是不同的，如下所示。

Listing 18. Create and Initialize Session Cipher and Hash
清单 18。创建并初始化会话密码和哈希

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;

sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA256;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey =
sampleCipherKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
sizeof(sampleCipherKey);
/* The resulting MAC is to be placed immediately after the ciphertext
*/ sessionSetupData.digestIsAppended = CPA_TRUE;
sessionSetupData.verifyDigest = CPA_FALSE;
```

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder = CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;

sessionSetupData.cipherSetupData.cipherAlgorithm = CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA256;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey = sampleCipherKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
sizeof(sampleCipherKey);
/* The resulting MAC is to be placed immediately after the ciphertext */
sessionSetupData.digestIsAppended = CPA_TRUE; sessionSetupData.verifyDigest =
CPA_FALSE;
```

When calling the function to perform the chained cipher and hash operation, some of the fields of the operational data structure are again different from the cipher case, as shown below.

当调用用函数来执行链式加密和哈希运算时，操作数据结构的一些字段再次与加密情况不同，如下所示。

Listing 19. Set up Operational Data Cipher and Hash
清单 19。设置操作数据加密和散列

```
/** Populate the structure containing the operational data that is
 * needed to run the algorithm
```

```
/** Populate the structure containing the operational data that is
 * needed to run the algorithm
```

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType =
CPA_CY_SYM_PACKET_TYPE_FULL; pOpData->pIv =
pIvBuffer;
pOpData->ivLenInBytes =
sizeof(sampleCipherIv); pOpData-
>cryptoStartSrcOffsetInBytes = 0;
```

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
pOpData->pIv = pIvBuffer;
pOpData->ivLenInBytes = sizeof(sampleCipherIv);
pOpData->cryptoStartSrcOffsetInBytes = 0;
pOpData->hashStartSrcOffsetInBytes = 0;
pOpData->messageLenToCipherInBytes = sizeof(sampleAlgChainingSrc); pOpData-
```

Notice the `digestIsAppended` is set in the session; therefore, the MAC is placed immediately after the region to hash, and the `pDigestResult` parameter of the operational data is ignored.

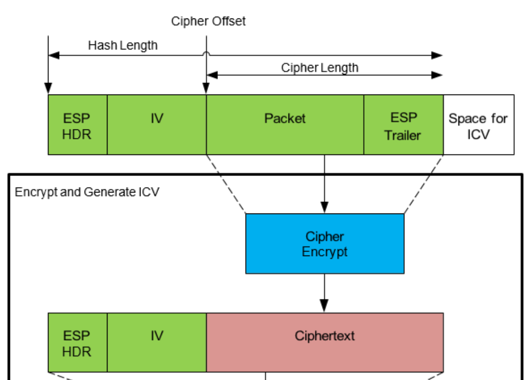
请注意，digestIsAppended 是在会话中设置的；因此，MAC 被放置在要散列的区域之后，并且操作数据的 pDigestResult 参数被忽略。

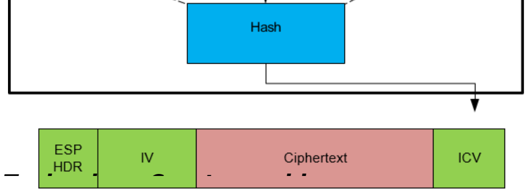
3.2.11 Chained Cipher and Hash – IPSec Like Use Case
3.2.12 链式密码和哈希 – 类似 IPSec 的用例

This example demonstrates the usage of the symmetric API for IPSec-like use cases, as described in Figure 5 and Figure 6. For the outbound direction, this example uses the symmetric API to perform a "chained" cipher and hash operation. It encrypts some plaintext using the Advanced Encryption Standard (AES) algorithm in CBC mode, and then performs a SHA1 HMAC operation on the ciphertext, initialization vector, and header, writing the Integrity Check Value (ICV) to the buffer immediately after the ciphertext. For the inbound direction, this example again uses the symmetric API to perform a "chained" hash and cipher operation. It performs a SHA1 HMAC operation on the ciphertext, initialization vector, and Header and compares the Result with the input ICV. Then it decrypts the ciphertext using the AES algorithm in CBC mode.

此示例演示了对称 API 在类似 IPSec 的用例中的用法，如中所述 Figure 5 Figure 6

Figure 5. IPSec Outbound
图 5。IPSec 出站




Hash
ESP HDR
IV
Ciphertext
ICV

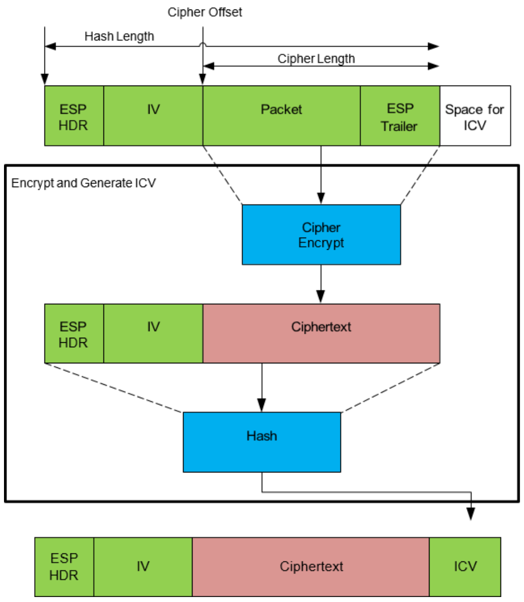
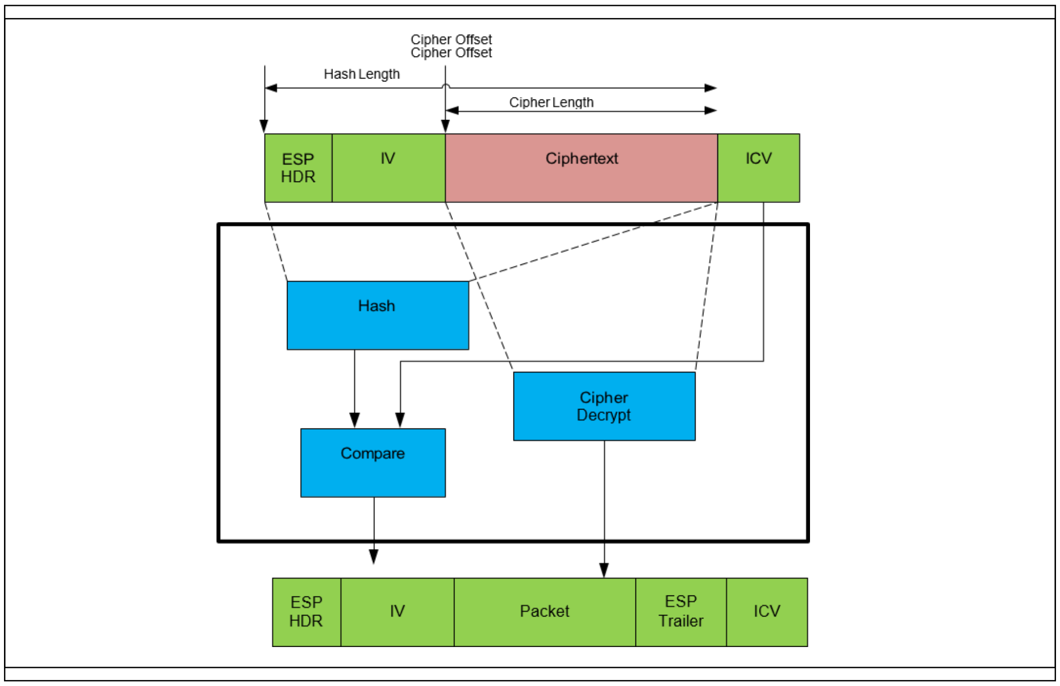

Cipher Offset
Hash Length
Cipher Length
ESP HDR
IV
Packet
ESP Trailer
Space for ICV
Encrypt and Generate ICV
Cipher Encrypt
ESP HDR
IV
Ciphertext
Hash
ESP HDR
IV
Ciphertext
ICV

Figure 6. IPSec Inbound
图 6。IPSec 入站



These samples are located in `/sym/ipsec_sample.`

这些示例位于/sym/ipsec_sample 中。

Again, only the differences compared to previous examples are highlighted:

同样，只突出显示了与前面示例相比的不同之处：

When creating and initializing a session in the outbound direction, the session initialization operational data structure is shown below.
创建和初始化出站方向的会话时，会话初始化操作数据结构如下所示。

Listing 20. Session Setup Data IPSec Outbound
清单 20。会话设置数据 IPSec 出站

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;

sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA1;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = ICV_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey =
```

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder = CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;

sessionSetupData.cipherSetupData.cipherAlgorithm = CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA1;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = ICV_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey = sampleAuthKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
sizeof(sampleAuthKey);
```

```
  /* Even though ICV follows immediately after the region to
                           hash
*   digestIsAppended is set to false in this case to
    workaround
 * errata number IXA00378322 */
sessionSetupData.digestIsAppended =
```

```
    /* Even though ICV follows immediately after the region to hash
*   digestIsAppended is set to false in this case to workaround
 * errata number IXA00378322 */
sessionSetupData.digestIsAppended = CPA_FALSE;
/* Generate the ICV in outbound direction */
```

When calling the function to perform the chained cipher and hash operation, the fields of the operational data structure are shown below.

当调用用函数执行链式密码和散列运算时，操作数据结构的字段如下所示。

Listing 21. Operational Data IPSec Outbound
清单 21。操作数据 IPSec 出站

In this example samplePayload is the packet data plus the Encapsulating Security Payload (ESP) trailer.

在此示例中，samplePayload 是数据包数据加上封装安全有效负载(ESP)尾部。

When creating and initializing a session in the inbound direction, the session initialization operational data structure is shown below.

当创建和初始化入站方向的会话时，会话初始化操作数据结构如下所示。

Listing 22. Session Setup Data IPSec Inbound
清单 22。会话设置数据 IPSec 入站

```
/* Populate the structure containing the operational data that is
 * needed to run the algorithm in outbound direction
*/ pOpData->sessionCtx = sessionCtx;
pOpData->packetType =
CPA_CY_SYM_PACKET_TYPE_FULL; pOpData->pIv =
pIvBuffer;
pOpData->ivLenInBytes = sizeof(sampleCipherIv);
pOpData->cryptoStartSrcOffsetInBytes =
sizeof(sampleEspHdrData) +
sizeof(sampleCipherIv);
pOpData->messageLenToCipherInBytes =
sizeof(samplePayload); pOpData->hashStartSrcOffsetInBytes
= 0;
pOpData->messageLenToHashInBytes =
sizeof(sampleEspHdrData) + sizeof(sampleCipherIv) +
sizeof(samplePayload);
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;
sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_CBC;
/* Even though ICV follows immediately after the region to hash
 * digestIsAppended is set to false in this case to workaround
 * errata number IXA00378322
```

```
/* Populate the structure containing the operational data that is
 * needed to run the algorithm in outbound direction */
pOpData->sessionCtx = sessionCtx;
pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
pOpData->pIv = pIvBuffer;
pOpData->ivLenInBytes = sizeof(sampleCipherIv); pOpData-
>cryptoStartSrcOffsetInBytes = sizeof(sampleEspHdrData) +
sizeof(sampleCipherIv);
pOpData->messageLenToCipherInBytes = sizeof(samplePayload); pOpData-
>hashStartSrcOffsetInBytes = 0;
pOpData->messageLenToHashInBytes =
sizeof(sampleEspHdrData) + sizeof(sampleCipherIv) +
sizeof(samplePayload);
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder = CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;
/* Even though ICV follows immediately after the region to hash
 * digestIsAppended is set to false in this case to workaround
sessionSetupData.cipherSetupData.cipherAlgorithm = CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
pOpData->pDigestResult = pSrcBuffer +
(sizeof(sampleEspHdrData) + sizeof(sampleCipherIv) + sizeof(samplePayload));
```

```
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT;

sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA1;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = ICV_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey =
sampleAuthKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
sizeof(sampleAuthKey);

/* ICV follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
/* Verify the ICV in the inbound direction */
```

```
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT;

sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA1;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = ICV_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey = sampleAuthKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
sizeof(sampleAuthKey);

/* ICV follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
/* Verify the ICV in the inbound direction */
sessionSetupData.verifyDigest = CPA_TRUE;
```

When calling the function to perform the chained hash and cipher operation, the fields of the operational data structure are listed below.

当调用函数来执行链式散列和密码运算时，操作数据结构的字段如下所列。

Listing 23. Operational Data IPSec Inbound

清单 23。操作数据 IPSec 入站

```
/** Populate the structure containing the operational data that is
 * needed to run the algorithm in inbound direction
*/ pOpData->sessionCtx = sessionCtx;
pOpData->packetType =
CPA_CY_SYM_PACKET_TYPE_FULL; pOpData->pIv =
pIvBuffer;
pOpData->ivLenInBytes =
sizeof(sampleCipherIv); pOpData-
>cryptoStartSrcOffsetInBytes =
    sizeof(sampleEspHdrData) +
sizeof(sampleCipherIv); pOpData-
>messageLenToCipherInBytes =
```

In the example above, bufferSize is the size of the data input (header, iv, ciphertext, and ICV).

在上面的例子中，bufferSize 是数据输入的大小(头、iv、密文和 ICV)。

```
/** Populate the structure containing the operational data that is
 * needed to run the algorithm in inbound direction */
pOpData->sessionCtx = sessionCtx;
pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
pOpData->pIv = pIvBuffer;
pOpData->ivLenInBytes = sizeof(sampleCipherIv);
pOpData->cryptoStartSrcOffsetInBytes =
    sizeof(sampleEspHdrData) + sizeof(sampleCipherIv);
pOpData->messageLenToCipherInBytes =
    bufferSize -
    (sizeof(sampleEspHdrData) + sizeof(sampleCipherIv) + ICV_LENGTH); pOpData-
>hashStartSrcOffsetInBytes = 0;
```

3.2.13 Chained Cipher and Hash – SSL Like Use Case

3.2.14 链式密码和哈希 – 类似 SSL 的用例

This example demonstrates the usage of the symmetric API for SSL-like use cases, as described in Figure 7 and Figure 8. For the outbound direction, this example employs the symmetric API to perform a "chained" hash and cipher operation. It performs a SHA1 HMAC2 operation on a sequence number, part of the header, and the plaintext.

该示例演示了对称 API 在类似 SSL 的用例中的用法，如 Figure 7 Figure 8

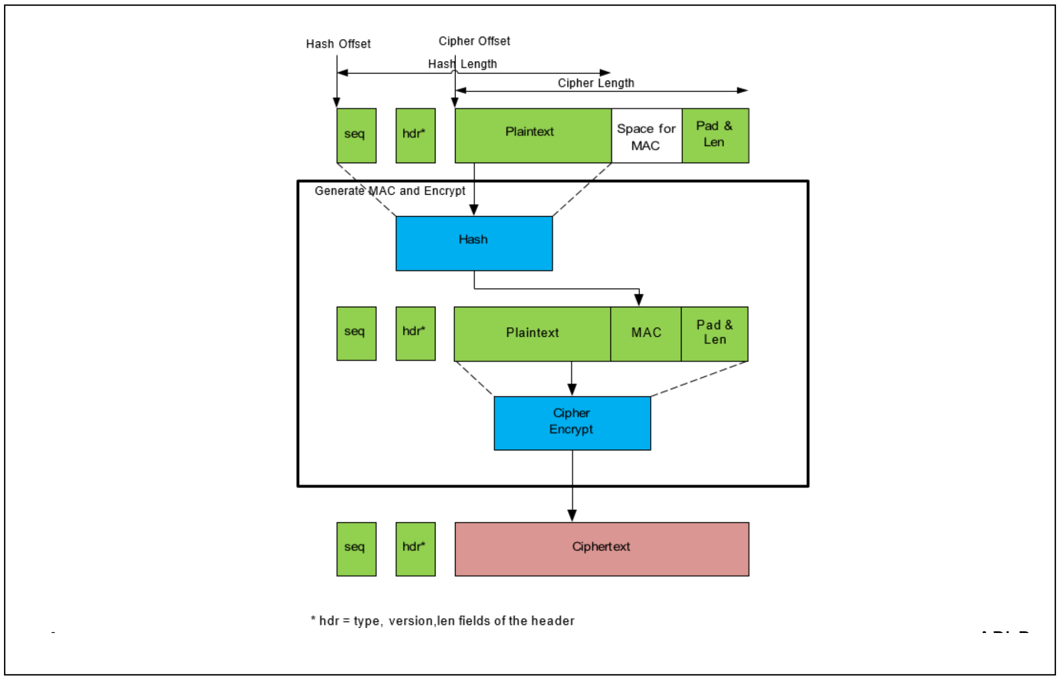
The resultant MAC is placed immediately after the plaintext. Then it encrypts the plaintext, MAC, and padding using the AES algorithm in CBC mode.[2]
生成的 MAC 将直接放在明文之后。然后，它在 CBC 模式下使用 AES 算法加密明文、MAC 和填充。[2]

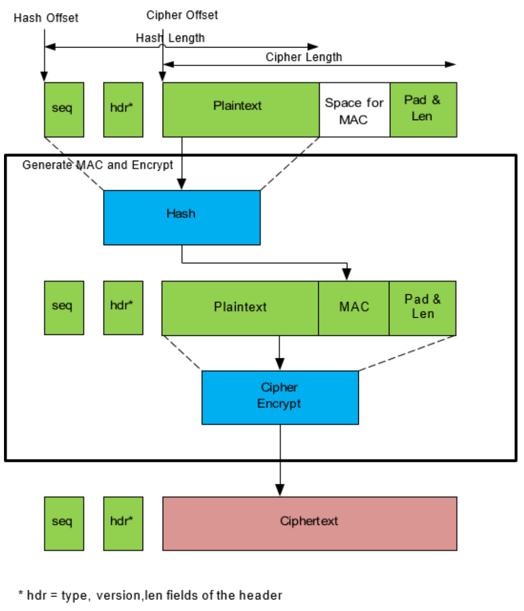
For the inbound direction, this example again employs the use of the symmetric API to perform a "chained" cipher and hash operation. It decrypts the ciphertext using the AES algorithm in CBC mode. Then it performs a SHA1 HMAC operation on the resultant plaintext, sequence number, and part of the Header and compares the Result with the input MAC.
对于入站方向，该示例再次使用对称 API 来执行“链式”加密和散列操作。它在 CBC 模式下使用 AES 算法解密密文。然后，它对生成的明文、序列号和部分报头执行 SHA1 HMAC 运算，并将结果与输入 MAC 进行比较。

Note: For the inbound direction to use the "chained" API, the length of the plaintext needs to be known before the ciphertext is decrypted to set `messageLenToHashInBytes` and the length field in the Header correctly.
注意：对于使用“链式”API 的入站方向，在解密密文之前需要知道明文的长度，以正确设置 messageLenToHashInBytes 和报头中的长度字段。

For stream ciphers (e.g., ARC4), there is no padding added in the outbound direction, so the length of the plaintext is simply the length of the ciphertext minus the length of the MAC. However, for block ciphers in CBC mode (as used in this example), the padlen is required to calculate the plaintext length. The final block of the ciphertext needs to be decrypted to discover the padlen. In this example, before calling the "chained" API, the final block of the ciphertext is decrypted to discover the padlen.
对于流密码(例如 ARC4)，在出站方向没有添加填充，因此明文的长度就是密文的长度减去 MAC 的长度。然而，对于 CBC 模式下的块密码(如本例中所用)，padlen 需要计算明文长度。需要解密密文的最后一个块来发现 padlen。在这个例子中，在调用“链式”API 之前，密文的最后一个块被解密以发现 padlen。

Figure 7. SSL Outbound
图 7。SSL 出站

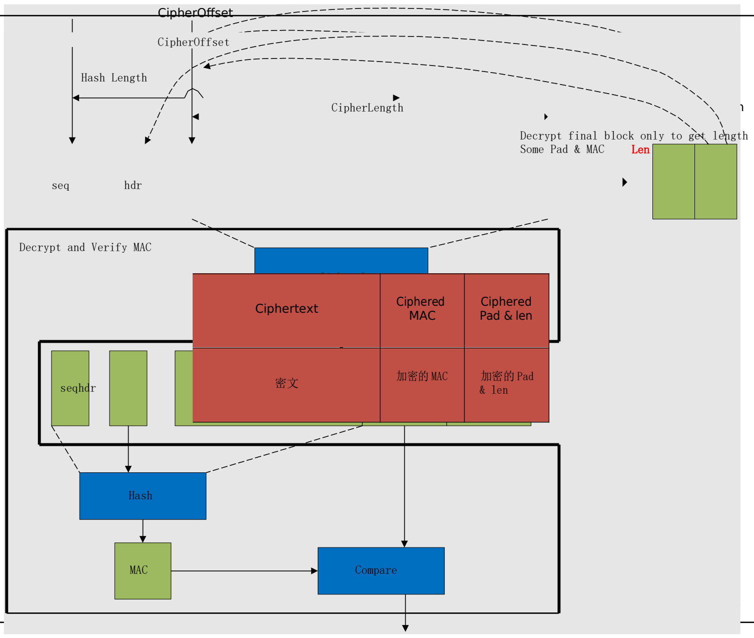




[2] *Not all SSL use cases use HMAC. For example, The Secure Sockets Layer (SSL) Protocol Version 3.0 (SSL)v3 (RFC 6106) does not use HMAC (in this case the nested hash functionality on the API can be used). However, The Transport Layer Security (TLS) Protocol (TLS) V1.2 (Table 1) , for example, does use the HMAC algorithm.*

2 不是所有的 SSL 用例都使用 HMAC。例如，安全套接字层(SSL)协议版本 3.0 (SSL)v3 (RFC 6106)不使用 HMAC(在这种情况下，可以使用 API 上的嵌套散列功能)。然而，传输层安全(TLS)协议(TLS)1.2 版(表 1)就使用了 HMAC 算法。

Figure 8. Secure Sockets Layer Inbound
图 8。安全套接字层入站



If using a block cipher in CBC mode, then the last ciphertext block is used as the IV for subsequent packets (or records) in Secure Sockets Layer (SSL) and TLSv1.0, whereas in TLSv1.1 and 1.2 an explicit IV is used. However, if using a stream cipher that does not use a synchronization vector (such as ARC4), the stream cipher state from the end of one packet is used to process the subsequent packets. If using the QA API in this case, then partial mode should be used to ensure the stream cipher state is maintained across multiple calls to the API.
如果在 CBC 模式下使用分组密码，则在安全套接字层(SSL)和 TLSv1.0 中，最后一个密文分组用作后续分组(或记录)的 IV，而在 TLSv1.1 和 1.2 中，使用显式 IV。但是，如果使用不使用同步向量的流

密码(如 ARC4)，则来自一个数据包末尾的流密码状态将用于处理后续数据包。如果在这种情况下使用 QA API，那么应该使用部分模式来确保在对 API 的多次调用中保持流密码状态。

Again, these examples are very similar to previous examples, so only the differences are highlighted:
同样，这些示例与前面的示例非常相似，因此只强调不同之处:

When creating and initializing a session in the outbound direction, the session setup data structure is shown below.
创建和初始化出站方向的会话时，会话设置数据结构如下所示。

Listing 24. Session Data SSL Outbound
清单 24。会话数据 SSL 出站

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;

sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
```

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder = CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;

sessionSetupData.cipherSetupData.cipherAlgorithm = CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA1;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = MAC_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey =
sampleAuthKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
sizeof(sampleAuthKey);

/* MAC follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
```

```
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = MAC_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey = sampleAuthKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
sizeof(sampleAuthKey);

/* MAC follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
/* Generate the MAC in outbound direction */
```

A buffer large enough to hold the plaintext, MAC and padding is required. The size of this buffer will be.

需要一个足够大的缓冲区来保存明文、MAC 和填充。这个缓冲区的大小是。

Listing 25. Buffer Size SSL Outbound
清单 25。SSL 出站缓冲区大小

```
bufferSize = sizeof(samplePayload) + MAC_LENGTH;
/* bufferSize needs to be rounded up to a multiple of the AES block
size */
padLen = 16 - bufferSize % 16;
bufferSize += padLen;
/* padLen excludes pad_length field */
```

```
bufferSize = sizeof(samplePayload) + MAC_LENGTH;
/* bufferSize needs to be rounded up to a multiple of the AES block size
*/
padLen = 16 - bufferSize % 16; bufferSize
+= padLen;
/* padLen excludes pad_length field */
```

This buffer is filled with plaintext and padding leaving room for the “chained” API operation to add the MAC.

这个缓冲区填充了明文和填充，为“链式”API 操作添加 MAC 留出了空间。

Listing 26. Buffer Setup SSL Outbound
清单 26。缓冲区设置 SSL 出站

```
memcpy(pSrcBuffer, samplePayload, sizeof(samplePayload));
/* Leave space for MAC but insert padding data. */
for (i = 0; i <= padLen; i++) {
    pSrcBuffer[(sizeof(samplePayload) + MAC_LENGTH + i)] = padLen;
```

```
memcpy(pSrcBuffer, samplePayload, sizeof(samplePayload));
/* Leave space for MAC but insert padding data */ for (i =
0; i <= padLen; i++) {
    pSrcBuffer[(sizeof(samplePayload) + MAC_LENGTH + i)] = padLen;
```

The session sequence number, the header and the buffer with the plaintext are described using a CpaBufferList:

使用 CpaBufferList 描述会话序列号、报头和具有明文的缓冲区。

Listing 27. BufferList Setup SSL Outbound
清单 27。缓冲列表设置 SSL 出站

```
pBufferList->pBuffers = pFlatBuffer;
pBufferList->numBuffers = numBuffers;
pBufferList->pPrivateMetaData =
pBufferMeta;

/* Seq number */
pFlatBuffer->dataLenInBytes =
SSL_CombinedHeadSize; pFlatBuffer->pData =
```

```
pBufferList->pBuffers = pFlatBuffer; pBufferList-
>numBuffers = numBuffers; pBufferList-
>pPrivateMetaData = pBufferMeta;

/* Seq number */
pFlatBuffer->dataLenInBytes = SSL_CombinedHeadSize;
pFlatBuffer->pData = pCombinedHeadBuffer; pFlatBuffer++;
memcpy((char *)pCombinedHeadBuffer + SESSION_SEQ_START
sizeof(sessSeqNum));
memcpy((char *)pCombinedHeadBuffer + HDR_START,
sampleHdrData,
sizeof(sampleHdrData));

/* Data */
pFlatBuffer->dataLenInBytes =
```

```
sizeof(sessSeqNum));
memcpy((char *)pCombinedHeadBuffer + HDR_START,
sampleHdrData,
sizeof(sampleHdrData));

/* Data */
pFlatBuffer->dataLenInBytes = bufferSize;
```

When calling the function to perform the chained hash and cipher operation, the fields of the operational data structure are shown below.

当调用函数执行链式散列和密码运算时，操作数据结构的字段如下所示。

Listing 28. Operational Data SSL Outbound

清单 28。运营数据 SSL 出站

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType =
CPA_CY_SYM_PACKET_TYPE_FULL; pOpData->pIv =
pIvBuffer;
pOpData->ivLenInBytes = sizeof(sampleCipherIv);
pOpData->cryptoStartSrcOffsetInBytes =
SSL_CombinedHeadSize; pOpData->messageLenToCipherInBytes =
bufferSize;
pOpData->hashStartSrcOffsetInBytes =
```

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
pOpData->pIv = pIvBuffer;
pOpData->ivLenInBytes = sizeof(sampleCipherIv);
pOpData->cryptoStartSrcOffsetInBytes = SSL_CombinedHeadSize; pOpData->messageLenToCipherInBytes = bufferSize;
pOpData->hashStartSrcOffsetInBytes = SESSION_SEQ_START;
pOpData->messageLenToHashInBytes = sizeof(sessSeqNum) + sizeof(sampleHdrData) +
bufferSize + MAC_LENGTH;
```

When creating and initializing a session in the inbound direction, the session setup data structure is shown below.

创建和初始化入站方向的会话时，会话设置数据结构如下所示。

Listing 29. Session Data SSL Inbound

清单 29。会话数据 SSL 入站

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;

sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA1;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = MAC_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey =
sampleAuthKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
sizeof(sampleAuthKey);
```

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder = CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;

sessionSetupData.cipherSetupData.cipherAlgorithm = CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA1;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = MAC_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey = sampleAuthKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
sizeof(sampleAuthKey);

/* MAC follows immediately after the region to hash */
```

```
sessionSetupData.digestIsAppended = CPA_TRUE;
/* Generate the MAC in outbound direction */
```

```
sessionSetupData.digestIsAppended = CPA_TRUE;
/* Generate the MAC in outbound direction */
```

In this case the length of the ciphertext is `bufferSize` to calculate the padLen the final block is decrypted.

在这种情况下，密文的长度被缓冲以计算最终块被解密的长度。

Listing 30. Calculating padLen SSL Inbound

清单 30。正在计算 padLen SSL 入站

```
Cpa8U resBuff[16];
/* For decrypt direction need to decrypt the final block
 * to determine the messageLenToHashInBytes */
status =
sampleCodeAesCbcDecrypt(sampleCipherKey,
sizeof(sampleCipherKey),
(pSrcBuffer + (bufferSize - 32)), /* IV */
(pSrcBuffer + (bufferSize - 16)), /* src */
resBuff);                           /* dest */
/* padLen is the last byte decrypted incremented by one to
```

When calling the function to perform the chained cipher and hash operation, the fields of the operational data structure are:

当调用用函数来执行链式密码和散列运算时，操作数据结构的字段是。

Listing 31. Operational Data (SSL Inbound)

清单 31。运营数据 SSL 入站

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType =
CPA_CY_SYM_PACKET_TYPE_FULL; pOpData->pIv =
pIvBuffer;
pOpData->ivLenInBytes = sizeof(sampleCipherIv);
pOpData->cryptoStartSrcOffsetInBytes =
SSL_CombinedHeadSize; pOpData->messageLenToCipherInBytes =
bufferSize;
pOpData->hashStartSrcOffsetInBytes = SESSION_SEQ_START;
pOpData->messageLenToHashInBytes = sizeof(sessSeqNum) + sizeof(sampleHdrData) +
bufferSize - MAC_LENGTH - padLen;
```

3.2.15 Chained Cipher and Hash - CCM Use Case

3.2.16 链式密码和哈希 - CCM 使用案例

This example demonstrates the usage of the symmetric API to perform a CCM operation as described in NIST publication SP800-38C (Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality, refer to Table 1).

该示例展示了对称 API 的使用，以执行如 NIST 出版物 SP800-38C 中所描述的 CCM 操作(分组密码操作模式的推荐:用于认证和机密性的 CCM 模式，参考 Table 1

This sample is located in `/sym/ccm_sample`

该示例位于/sym/ccm_sample 中

The example is very similar to the cipher example, so only the differences are highlighted:

该示例与密码示例非常相似，因此只突出显示不同之处:

For the generation-encryption process the session setup data is shown below:
对于生成加密过程，会话设置数据如下所示：

Listing 32. Session Data CCM Generate-Encrypt
清单 32。会话数据 CCM 生成-加密

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;
sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_CCM;
sessionSetupData.cipherSetupData.pCipherKey = sampleKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleKey); sessionSetupData.cipherSetupData.cipherDirection =
    CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_AES_CCM;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
    /* Note that for CCM authKey and authKeyLen are not required this
                 * information is provided by the cipherKey in
                                                  cipherSetupData
 */
sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes =
    sizeof(sampleAssocData);
/* For CCM digestAppended and digestVerify are not required in
 * the encrypt direction digestAppended is CPA_TRUE and
 * digestVerify is CPA_FALSE
 */
```

For the decryption-verification process the session setup data is:
对于解密-验证过程，会话建立数据是：

Listing 33. Session Data CCM Decrypt-Verify
清单 33。会话数据 CCM 解密-验证

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;
sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_CCM;
sessionSetupData.cipherSetupData.pCipherKey = sampleKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleKey); sessionSetupData.cipherSetupData.cipherDirection =
    CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_AES_CCM;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes =
    sizeof(sampleAssocData);
/* For CCM digestAppended and digestVerify are not required. In
```

The IV and AAD buffers are allocated as shown below:
IV 和 AAD 缓冲器的分配如下所示:

Listing 34. CCM Allocate IV and AAD Buffers
清单 34。CCM 分配 IV 和 AAD 缓冲区

```
        /* Allocate memory to store IV. For CCM this is the counter block
         * ctr0 (size equal to AES block size). The implementation will
         * construct the ctr0 block given the nonce. Space for ctr0 must be
         * allocated here. */
            status = PHYS_CONTIG_ALLOC(&pIvBuffer, AES_BLOCK_SIZE);
}
if (CPA_STATUS_SUCCESS == status) {
        /* Allocate memory for AAD. For CCM this memory will hold the 16 byte
         * B0 block, the 2 bytes encoded length of associated data, the
         * associated data itself and any padding to ensure total size is
         * a multiple of the AES block size
         */
        aadBuffSize = B0_BLOCK_SIZE + ALEN_ENCODING_SIZE +
sizeof(sampleAssocData);
        if (aadBuffSize % AES_BLOCK_SIZE) {
            aadBuffSize += AES_BLOCK_SIZE - (aadBuffSize % AES_BLOCK_SIZE);
        }
```

The operational data needed to perform the generate-encrypt or decrypt-verify operation is shown below:
执行生成-加密或解密-验证操作所需的操作数据如下所示:

Listing 35. CCM Operational Data
清单 35。CCM 操作数据

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
pOpData->pIv = pIvBuffer;
/* Even though the iv buffer is 16 bytes the ivLenInBytes
 * is set to the length of the nonce. For CCM valid lengths
 * are in the range 7-13
 */
pOpData->ivLenInBytes = sizeof(sampleNonce);
pOpData->cryptoStartSrcOffsetInBytes = 0;
pOpData->messageLenToCipherInBytes = sizeof(samplePayload);
/* Notice for CCM hash offset and length are not required */
pOpData->pAdditionalAuthData = pAadBuffer;
/* Populate pIv and pAdditionalAuthData buffers with nonce and assoc data
*/
CPA_CY_SYM_CCM_SET_NONCE(pOpData, sampleNonce, sizeof(sampleNonce));
CPA_CY_SYM_CCM_SET_AAD(pOpData, sampleAssocData,
```

3.2.17 Chained Cipher and Hash – GCM Use Case
3.2.18 链式密码和哈希 – GCM 用例

This example demonstrates the usage of the symmetric API to perform a GCM operation as described in *NIST publication SP800-38D (Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, refer to Table 1).

这个例子演示了对称 API 的使用，以执行如 NIST 出版物 SP800-38D(分组密码操作模式的推荐:伽罗瓦/计数器模式(GCM)和 GMAC，参考 Table 1

These samples are located in `/sym/gcm_sample`
这些示例位于/sym/gcm_sample 中

An example of the session setup data and operational data for GCM authenticated encryption and decryption is shown below.

GCM 认证加密和解密的会话设置数据和操作数据的示例如下所示。

For authenticated encryption the session setup data is:

对于认证加密，会话设置数据为:

Listing 36. Session Data GCM Auth-Encrypt
清单 36。会话数据 GCM 授权加密

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;
sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_GCM;
sessionSetupData.cipherSetupData.pCipherKey = sampleKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleKey); sessionSetupData.cipherSetupData.cipherDirection =
    CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_AES_GCM;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = TAG_LENGTH;
/* For GCM authKey and authKeyLen are not required this information
 * is provided by the cipherKey in cipherSetupData */
sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes
=
    sizeof(sampleAddAuthData);
/* Tag follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
```

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;
sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_GCM;
sessionSetupData.cipherSetupData.pCipherKey = sampleKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleKey); sessionSetupData.cipherSetupData.cipherDirection =
    CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT;
```

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder = CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;
sessionSetupData.cipherSetupData.cipherAlgorithm = CPA_CY_SYM_CIPHER_AES_GCM;
sessionSetupData.cipherSetupData.pCipherKey = sampleKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleKey);
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder = CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_AES_GCM;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = TAG_LENGTH;
/* For GCM authKey and authKeyLen are not required this information
sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes =
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_AES_GCM;
/* Tag follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
/* digestVerify is not required to be set. For GCM authenticated
 * encryption this value is understood to be CPA_FALSE */
```

```
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = TAG_LENGTH;

/* For GCM authKey and authKeyLen are not required this information
 * is provided by the cipherKey in cipherSetupData */
sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes
=
    sizeof(sampleAddAuthData);
/* Tag follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
```

```
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = TAG_LENGTH;

/* For GCM authKey and authKeyLen are not required this information
 * is provided by the cipherKey in cipherSetupData */
sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes =
    sizeof(sampleAddAuthData);
/* Tag follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
/* digestVerify is not required to be set. For GCM authenticated
```

For authenticated decryption the session setup data is:

对于认证解密，会话设置数据为:

Listing 37. Session Data GCM Auth-Decrypt
清单 37。会话数据 GCM 授权解密

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;

sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_GCM;
sessionSetupData.cipherSetupData.pCipherKey = sampleKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleKey); sessionSetupData.cipherSetupData.cipherDirection =
    CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_AES_GCM;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = TAG_LENGTH;
/* For GCM authKey and authKeyLen are not required this information
 * is provided by the cipherKey in cipherSetupData */
sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes
=
    sizeof(sampleAddAuthData);
/* Tag follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
```

```
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder = CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;

sessionSetupData.cipherSetupData.cipherAlgorithm = CPA_CY_SYM_CIPHER_AES_GCM;
sessionSetupData.cipherSetupData.pCipherKey = sampleKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleKey);
sessionSetupData.cipherSetupData.cipherDirection =
    CPA_CY_SYM_CIPHER_DIRECTION_DECRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_AES_GCM;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = TAG_LENGTH;
/* For GCM authKey and authKeyLen are not required this information
 * is provided by the cipherKey in cipherSetupData */
sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes =
    sizeof(sampleAddAuthData);
/* Tag follows immediately after the region to hash */
sessionSetupData.digestIsAppended = CPA_TRUE;
/* digestVerify is not required to be set. For GCM authenticated
 * decryption this value is understood to be CPA_TRUE */
```

The IV and AAD buffers are allocated as shown below:

IV 和 AAD 缓冲器的分配如下所示:

Listing 38. GCM Allocate IV and AAD Buffers
清单 38。GCM 分配 IV 和 AAD 缓冲区

```
    /* Allocate memory to store IV. For GCM this is the block J0
     * (size equal to AES block size). If iv is 12 bytes the
     * implementation will construct the J0 block given the iv.
     * If iv is not 12 bytes then the user must construct the J0
```

```
    /* Allocate memory to store IV. For GCM this is the block J0
     * (size equal to AES block size). If iv is 12 bytes the
     * implementation will construct the J0 block given the iv.
     * If iv is not 12 bytes then the user must construct the J0
```

```
         * must be allocated. */
        status = PHYS_CONTIG_ALLOC(&pIvBuffer, AES_BLOCK_SIZE);
    }
    if (CPA_STATUS_SUCCESS == status) {
        /* Allocate memory for AAD. For GCM this memory will hold the
         * additional authentication data and any padding to ensure total
         * size is a multiple of the AES block size
        */ aadBuffSize = sizeof(sampleAddAuthData);
        if (aadBuffSize % AES_BLOCK_SIZE) {
            aadBuffSize += AES_BLOCK_SIZE - (aadBuffSize % AES_BLOCK_SIZE);
        }
        status = PHYS_CONTIG_ALLOC(&pAadBuffer, aadBuffSize);
```

```
         * must be allocated. */
        status = PHYS_CONTIG_ALLOC(&pIvBuffer, AES_BLOCK_SIZE);
    }
    if (CPA_STATUS_SUCCESS == status) {
        /* Allocate memory for AAD. For GCM this memory will hold the
         * additional authentication data and any padding to ensure total
         * size is a multiple of the AES block size */
        aadBuffSize = sizeof(sampleAddAuthData);
        if (aadBuffSize % AES_BLOCK_SIZE) {
            aadBuffSize += AES_BLOCK_SIZE - (aadBuffSize % AES_BLOCK_SIZE);
        }
        status = PHYS_CONTIG_ALLOC(&pAadBuffer, aadBuffSize);
```

The operational data needed to perform the encrypt or decrypt operation is:

执行加密或解密操作所需的操作数据是:

Listing 39. GCM Operational Data
清单 39。GCM 操作数据

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType =
CPA_CY_SYM_PACKET_TYPE_FULL; pOpData->pIv =
pIvBuffer;
/* In this example iv is 12 bytes. The implementation
 * will use the iv to generation the J0 block
 */
memcpy(pIvBuffer, sampleIv,
sizeof(sampleIv)); pOpData->ivLenInBytes =
sizeof(sampleIv); pOpData-
>cryptoStartSrcOffsetInBytes = 0;
```

```
pOpData->sessionCtx = sessionCtx;
pOpData->packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
pOpData->pIv = pIvBuffer;
/* In this example iv is 12 bytes. The implementation
 * will use the iv to generation the J0 block
 */
memcpy(pIvBuffer, sampleIv, sizeof(sampleIv));
pOpData->ivLenInBytes = sizeof(sampleIv); pOpData->cryptoStartSrcOffsetInBytes = 0;
pOpData->messageLenToCipherInBytes = sizeof(samplePayload);
/* For GCM hash offset and length are not required */
```

GMAC is supported using the same API and similar data structures as the general GCM case shown above. However, for GMAC, the `messageLenToCipherInBytes` will be set to **0**.

使用与上面显示的一般 GCM 情况相同的 API 和相似的数据结构来支持 GMAC。但是，对于 GMAC，messageLenToCipherInBytes 将被设置为 0。

3.2.19 Chained Cipher and Hash Using the Symmetric Data Plane API

3.2.20 使用对称数据平面 API 的链式密码和散列

This example demonstrates the usage of the data plane symmetric API to perform a “chained” cipher and hash operation. It encrypts some sample text using the AES-256 algorithm in CBC mode, and then performs an SHA-256 HMAC operation on the ciphertext, writing the MAC to the buffer immediately after the ciphertext.

此示例演示了如何使用数据平面对称 API 来执行“链式”加密和哈希运算。它在 CBC 模式下使用 AES-256 算法加密一些样本文本，然后对密文执行 SHA-256 HMAC 运算，在密文后立即将 MAC 写入缓冲区。

This example has been simplified to demonstrate the basics of how to use the API and build the structures required. This example does not demonstrate the optimal way to use the API to get the maximum performance for a particular implementation. Refer to Implementation Specific Documentation in Table 1 (for example, the Intel® Communications Chipset 8900 to 8920 Series Software Programmer’s Guide) and performance sample code for a guide on how to use the API for best performance.
这个例子已经被简化，以演示如何使用 API 和构建所需结构的基础知识。这个例子没有展示使用 API 来获得特定实现的最佳性能的最佳方式。请参考中的实施特定文档 Table 1

These samples are located in `/sym/symdp_sample`
这些示例位于/sym/symdp_sample 中

Note: Use of the data plane symmetric API follows some of the same basic steps as the traditional symmetric API:
注意:数据平面对称 API 的使用遵循一些与传统对称 API 相同的基本步骤:

- Discover and start up the cryptographic service instance.
- 发现并启动加密服务实例。
- Register a callback function for the instance.
- 为实例注册一个回调函数。
- Create and initialize a session.
- 创建并初始化会话。
- Enqueue the symmetric operation on the instance.
- 将实例上的对称操作排入队列。
- Submit the symmetric operation for processing.
- 提交对称操作进行处理。
- Poll the instance for a response.
- 轮询实例以获得响应。
- Tear down the session.
- 拆除会话。
- Stop the Cryptographic service instance.
- 停止加密服务实例。

The following are the steps in more detail:
以下是更详细的步骤:

- Cryptographic service instances are discovered and started in the same way and using the same API as the traditional symmetric use cases described in Listing 4, Listing 5, and, Listing 6.
- 发现和启动加密服务实例的方式和使用的 API 与中描述的传统对称用例相同 Listing 4Listing 5Listing 6
- The next step is to register a callback function for the cryptographic instance:
- 下一步是为加密实例注册一个回调函数:

The function is called back in the context of the polling function when an asynchronous operation has completed. This function can perform whatever processing is appropriate to the Application.
当异步操作完成时，在轮询函数的上下文中回调该函数。这个函数可以执行任何适合应用程序的处理。

Callback differs from the traditional symmetric API, where the callback function is registered for the session.
回调不同于传统的对称 API，在传统的对称 API 中，回调函数是为会话注册的。

Listing 40. Register Callback Function
清单 40。注册回调函数

```
status = cpaCySymDpRegCbFunc(cyInstHandle, symDpCallback);
```

```
status = cpaCySymDpRegCbFunc(cyInstHandle, symDpCallback);
```

Create and initialize a session:
创建并初始化会话:

Listing 41. Create and Initialize Data Plane Session
清单 41。创建并初始化数据平面会话

```
sessionSetupData.sessionPriority = CPA_CY_PRIORITY_HIGH;
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;

sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
    CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
```

```
sessionSetupData.sessionPriority = CPA_CY_PRIORITY_HIGH;
sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
sessionSetupData.algChainOrder = CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;

sessionSetupData.cipherSetupData.cipherAlgorithm = CPA_CY_SYM_CIPHER_AES_CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes = sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
    CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA256;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
session setup data . hashsetup data . HASH MODE = CPA _ CY _ SYM _ HASH _ MODE _ AUTH;
session setup data . hashsetup data . digestresultleninbytes = DIGEST _ LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey =
sampleCipherKey;
sessionsetupdata . hashsetupdata . authmodesetupdata . authkey = sampleCipherKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
    sizeof(sampleCipherKey);
sessionsetupdata . hashsetupdata . authmodesetupdata . authkeyleninbytes =
    sizeof(sampleCipherKey);


/* Even though MAC follows immediately after the region to hash
                  /*即使 MAC 紧跟在要散列的区域之后
*   digestIsAppended is set to false in this case to workaround
*   在这种情况下，digestIsAppended 设置为 false 以解决问题
 * errata number IXA00378322 */
sessionSetupData.digestIsAppended = CPA_FALSE;
sessionSetupData.verifyDigest = CPA_FALSE;
 * 勘误表编号 ixa 00378322 */session setup data .
digestisappend = CPA _ FALSE; session setup data .
verify digest = CPA _ FALSE;


/* Determine size of session context to allocate */
PRINT_DBG("cpaCySymDpSessionCtxGetSize\n");
/*确定要分配的会话上下文的大小*/PRINT _ DBG(" cpacysymdpsessiontxgetsize \ n ");
status = cpaCySymDpSessionCtxGetSize(cyInstHandle,
status = cpacysymdpsessiontxgetsize(cyInstHandle,
                                     &sessionSetupData,
                                     &sessionCtxSize);
                                     &sessionSetupData, &
                                     session ctxsize);
}
}


if (CPA_STATUS_SUCCESS == status) {
if(CPA _ STATUS _ SUCCESS = = STATUS){
    /* Allocate session context */
    /*分配会话上下文*/
    status = PHYS_CONTIG_ALLOC(&sessionCtx, sessionCtxSize);
    status = PHYS _重叠_分配(&sessionCtx, session ctxsize);
}
}
```

```
if (CPA_STATUS_SUCCESS == status) {
if(CPA _ STATUS _ SUCCESS = = STATUS){
    /* Initialize the session */ PRINT_DBG("cpaCySymDpInitSession\
    n");
    /*初始化会话*/PRINT _ DBG(″ cpaCySymDpInitSession \ n ″);
    status = cpaCySymDpInitSession(cyInstHandle, &sessionSetupData,
sessionCtx);
    status = cpaCySymDpInitSession(cyInstHandle, &sessionSetupData, session CTX);
}
}


#ifdef LAC_HW_PRECOMPUTES
# ifdef LAC _ HW _预计算
if (CPA_STATUS_SUCCESS == status) {
if(CPA _ STATUS _ SUCCESS = = STATUS){
    /* Poll for hw pre-compute responses. */
    do {
    /*轮询硬件预计算响应。*/ do {
        status = icp_sal_CyPollDpInstance(cyInstHandle, 0);
        status = ICP _ sal _ CyPollDpInstance(cyInstHandle, 0);
    } while (CPA_STATUS_SUCCESS != status);
    } while (CPA_STATUS_SUCCESS! =状态);
}
}
#endif
#endif
```

In this example, data is stored in flat buffers (as opposed to scatter gather lists). The operational data in this case is shown below.
在本例中，数据存储在平面缓冲区中(与分散收集列表相反)。这种情况下的操作数据如下所示。

Listing 42. Data Plane Operational Data
清单 42。数据平面操作数据

```
sessionSetupData.sessionPriority = CPA_CY_PRIORITY_HIGH;
sessionSetupData.sessionPriority = CPA_CY_PRIORITY_HIGH;
sessionSetupData.algChainOrder =
CPA_CY_SYM_ALG_CHAIN_ORDER_CIPHER_THEN_HASH;
session setup data . ALG CHAIN ORDER = CPA _ CY _ SYM _ ALG _ CHAIN _ ORDER _ CIPHER _
THEN _ HASH;


sessionSetupData.cipherSetupData.cipherAlgorithm =
CPA_CY_SYM_CIPHER_AES_CBC;
session setup data . CIPHER setup data . CIPHER algorithm = CPA _ CY _ SYM _ CIPHER _ AES _
CBC;
sessionSetupData.cipherSetupData.pCipherKey = sampleCipherKey;
session setup data . cipher setup data . PCI pher key = sample cipher key;
sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
sizeof(sampleCipherKey);
sessionsetupdata . ciphersetupdata . cipherkeyleninbytes = sizeof(sampleCipherKey);
sessionSetupData.cipherSetupData.cipherDirection =
    CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
session setup data . CIPHER setup data . CIPHER DIRECTION = CPA _ CY _ SYM _ CIPHER _
    DIRECTION _ ENCRYPT;


sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA256;
sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
session setup data . hashsetup data . hashalgorithm = CPA _ CY _ SYM _哈希_ SHA256session
setup data . hashsetup data . HASH MODE = CPA _ CY _ SYM _ HASH _ MODE _ AUTH; session
setup data . hashsetup data . digestresultleninbytes = DIGEST _ LENGTH;
sessionSetupData.hashSetupData.authModeSetupData.authKey =
sampleCipherKey;
sessionsetupdata . hashsetupdata . authmodesetupdata . authkey = sampleCipherKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
    sizeof(sampleCipherKey);
sessionsetupdata . hashsetupdata . authmodesetupdata . authkeyleninbytes =
    sizeof(sampleCipherKey);


/* Even though MAC follows immediately after the region to hash
                  /*即使 MAC 紧跟在要散列的区域之后
*   digestIsAppended is set to false in this case to workaround
*   在这种情况下，digestIsAppended 设置为 false 以解决问题
 * errata number IXA00378322 */
sessionSetupData.digestIsAppended = CPA_FALSE;
sessionSetupData.verifyDigest = CPA_FALSE;
 * 勘误表编号 ixa 00378322 */session setup data .
digestisappend = CPA _ FALSE; session setup data .
verify digest = CPA _ FALSE;
```

```
/* Determine size of session context to allocate */
PRINT_DBG("cpaCySymDpSessionCtxGetSize\n");

/*确定要分配的会话上下文的大小*/PRINT _ DBG(″ cpacysymdpsessiontxgetsize \ n ″);
status = cpaCySymDpSessionCtxGetSize(cyInstHandle,
status = cpacysymdpsessiontxgetsize(cyInstHandle,
                                                  &sessionSetupData,
                                                  &sessionCtxSize);

                                                  &sessionSetupData, &
                                                  session ctxsize);
}
}


if (CPA_STATUS_SUCCESS == status) {
if(CPA _ STATUS _ SUCCESS = = STATUS){
    /* Allocate session context */
    /*分配会话上下文*/
    status = PHYS_CONTIG_ALLOC(&sessionCtx, sessionCtxSize);
    status = PHYS _重叠_分配(&sessionCtx, session ctxsize);
}
}


if (CPA_STATUS_SUCCESS == status) {
if(CPA _ STATUS _ SUCCESS = = STATUS){
    /* Initialize the session */ PRINT_DBG("cpaCySymDpInitSession\
    n");
    /*初始化会话*/PRINT _ DBG(″ cpaCySymDpInitSession \ n ″);
    status = cpaCySymDpInitSession(cyInstHandle, &sessionSetupData,
sessionCtx);
    status = cpaCySymDpInitSession(cyInstHandle, &sessionSetupData, session CTX);
}
}


#ifdef LAC_HW_PRECOMPUTES
# ifdef LAC _ HW _预计算
if (CPA_STATUS_SUCCESS == status) {
if(CPA _ STATUS _ SUCCESS = = STATUS){
    /* Poll for hw pre-compute responses. */
    /*轮询硬件预计算响应。*/
```

```
        do {
            status = icp_sal_CyPollDpInstance(cyInstHandle, 0);
        } while (CPA_STATUS_SUCCESS != status);
    }
```

```
        do {
            status = icp_sal_CyPollDpInstance(cyInstHandle, 0);
        } while (CPA_STATUS_SUCCESS != status);
    }
```

This request is then enqueued on the instance.

然后，该请求在实例上排队。

Listing 43.　Data Plane Enqueue

清单 43。数据平面排队

Other requests can now be enqueued before submitting all the requests to be processed. Enqueued requests allow the cost of submitting a request (which can be expensive, in terms of cycles, for some hardware-based implementations) to be amortized over all enqueued requests on the instance. Once sufficient requests have been enqueued they are all submitted for processing.

现在，在提交所有要处理的请求之前，可以将其他请求排队。入队请求允许提交请求的成本(对于一些基于硬件的实现来说，在周期方面可能是昂贵的)分摊到实例上的所有入队请求上。一旦有足够多的请求进入队列，它们都会被提交进行处理。

Listing 44.　Data Plane Perform

清单 44。数据平面执行

- An alternative to calling the `cpaCySymDpPerformOpNow` function is to set `performOpNow` to `CPA_TRUE` when calling the enqueue functions (`cpaCySymDpEnqueueOp` or `cpaCySymDpEnqueueOpBatch`). This is illustrated in Sections 4.4, Sample – Data Compression Data Plane API example.
- 调用 cpaCySymDpPerformOpNow 函数的另一种方法是在调用入队函数(cpaCySymDpEnqueueOp 或 cpaCySymDpEnqueueOpBatch)时将 PerformOpNow 设置为 CPA_TRUE。这将在部分中进行说明 4.4Sample – Data Compression Data Plane API
- After submitting several requests and possibly doing other work (e.g., enqueuing and submitting more requests), the Application can poll for responses that invoke the callback function registered with the instance. Refer to Table 1 Implementation Specific Documentation for information on the implementations polling functions.
- 在提交了几个请求并可能做了其他工作(例如，排队并提交更多请求)之后，应用程序可以轮询调用向实例注册的回调函数的响应。涉及 Table 1
- Once all requests associated with a session have been completed, the session can be removed.
- 一旦完成了与会话相关联的所有请求，就可以删除该会话。

Listing 45. Wait for Outstanding Requests
清单 45。等待未完成的请求

Since cryptographic API v2.2 before removing the symmetric session context, it is recommended to wait for the completion of any outstanding request using `cpaCySymSessionInUse`.

由于加密 API v2.2 在删除对称会话上下文之前，建议使用 cpaCySymSessionInUse 等待任何未完成的请求完成。

It is executed in the `symSessionWaitForInflightReq` call, which polls for the in-flight requests.

它在 symSessionWaitForInflightReq 调用中执行，该调用轮询进行中的请求。

Listing 46. Data Plane Remove Session
清单 46。数据平面删除会话

Since Cryptographic API v2.2, two new functions have been implemented:

自加密 API v2.2 版以来，实现了两个新功能:

- The function `cpaCySymUpdateSession` can be used to update certain parameters of a session like a cipher key, the cipher direction, and the authentication key. `cpaCySymSessionInUse` indicates whether there are outstanding requests on a given session.
- 函数 cpaCySymUpdateSession 可用于更新会话的某些参数，如加密密钥、加密方向和身份验证密钥。cpaCySymSessionInUse 指示给定会话上是否有未完成的请求。

As a result of the implementation of this feature, the behavior of `cpaCySymRemoveSession` has been changed. The `cpaCySymRemoveSession` fails if there is an outstanding request for the session that the user is trying to remove. 由于实现了此功能，cpaCySymRemoveSession 的行为已被更改。如果用户试图删除的会话有未完成的请求，则 cpaCySymRemoveSession 将失败。

- As a result, it is recommended to wait for the completion of any outstanding request, using `cpaCySymSessionInUse`, before removing a session.
- 因此，建议在删除会话之前，使用 cpaCySymSessionInUse 等待任何未完成的请求完成。

3.2.21 TLS Key and MGF Mask Generation

3.2.22 TLS 密钥和 MGF 掩码生成

Refer Table 1 to the API manuals for full details of Key and Mask Generation operations.

参考 Table 1

1. Define a Flat Buffer callback function as per the API prototype, Refer Table 1 to the API manuals.
2. 根据 API 原型定义平面缓冲区回调函数，请参考 Table 1

   If synchronous operation is preferred, instead simply pass NULL to the API for the callback parameter.

   如果同步操作是首选，那么只需将 NULL 传递给回调参数的 API。
3. Allocate memory for the operation.
4. 为操作分配内存。
5. Populate data for the appropriate operation data structure, Refer Table 1 to the API manuals.
6. 为适当的操作数据结构填充数据，请参考 Table 1

   a. Fill in the Flat Buffers, a pointer to data, and length.
   b. 填写平面缓冲区、数据指针和长度。

c. Fill in the options for the operation required.
d. 填写所需操作的选项。

7. Call the appropriate key or Mask Generation API.
8. 调用适当的密钥或掩码生成 API。
9. Complete the operation.
10. 完成操作。

The API for TLS key operations is based on the Transport Layer Security (TLS) Protocol Version 1.1 standard, RFC 4346, Backward compatibility is supported with the legacy Transport Layer Security (TLS) Protocol Version 1.0 standard, RFC 2246, refer to Table 1. The user-defined label should be used for backward compatibility with the client write key, server write key, and iv block. Refer to Table 1 Intel® QAT Cryptographic API Reference Manual for details of populating `CpaCyKeyGenTlsOpData`, the operation data structure.

TLS 密钥操作的 API 基于传输层安全(TLS)协议版本 1.1 标准 RFC 4346，支持向后兼容传统传输层安全(TLS)协议版本 1.0 标准 RFC 2246，请参考 Table 1Table 1

The following sections describe examples of the parameter mapping to the Cryptographic API.
以下部分描述了映射到加密 API 的参数的示例。

3.2.11.1 Setting CpaCyKeyGenTlsOpData Structure Fields
3.2.11.2 设置 CpaCyKeyGenTlsOpData 结构字段

The Transport *Layer Security (TLS) Protocol Version 1.1 standard*, RFC 4346, refer to Table 1, Section 6.3 `key_block` is described as:

```
key_block =
PRF(SecurityParameters.master_se
cret, "key expansion",
```

传输层安全(TLS)协议版本 1.1 标准 RFC 4346 参考 Table 1

This maps to the Cryptographic API's `CpaCyKeyGenTlsOpData` as follows:

```
key_block = PRF(SecurityParameters.master_secret, "key.
expansion",
SecurityParameters.server_random +
```

这映射到加密 API 的 CpaCyKeyGenTlsOpData，如下所示:

TLS Key-Material Derivation:

TLS 密钥-材料推导:

```
tlsOp = CPA_CY_KEY_TLS_OP_KEY_MATERIAL_DERIVE
tlsOp = CPA _ CY _ KEY _ TLS _ OP _ KEY _ MATERIAL _ DERIVE
```

```
secret = master secret key
```

secret =主密钥

```
seed = server_random + client_random
userLabel = NULL
```

seed =服务器随机+客户端随机用户标签=空

Setting `CpaCyKeyGenTlsOpData` Structure Fields for Backward Compatibility.

设置 CpaCyKeyGenTlsOpData 结构字段以实现向后兼容。

1. In the *Transport Layer Security (TLS) Protocol Version 1.0 standard*, RFC 2246, refer to Table 1, Section 6.3 `final_client_write_key` is described as:

```
final_client_write_key = PRF(client_write_key,
                             "client write key",
```

2. 在传输层安全(TLS)协议版本 1.0 标准 RFC 2246 中，请参考 Table 1

final_client_write_key = PRF(client_write_key,
"client write key",

This maps to the Cryptographic API's `CpaCyKeyGenTlsOpData` as follows:

这映射到加密 API 的 CpaCyKeyGenTlsOpData，如下所示:

```
TLS User Defined Derivation:
```

TLS 用户定义的派生:

```
tlsOp = CPA_CY_KEY_TLS_OP_USER_DEFINED
```

tlsOp = CPA _ CY _ KEY _ TLS _ OP _用户定义

```
secret = client_write_key
```

secret =客户端写入密钥

```
seed = client_random + server_random
userLabel = "client write key"
```

seed = client _ random+server _ random user label = ″客户端写入密钥″

3. In the *Transport Layer Security (TLS) Protocol v1.0 standard*, RFC 2246, refer to Table 1, Section 6.3 `final_server_write_key` is described as:

```
final_server_write_key = PRF(server_write_key,
                             "server write key",
                             client_random + server_random)
```

4. 在传输层安全(TLS)协议 1.0 版标准 RFC 2246 中，请参考 Table 1

final_server_write_key = PRF(server_write_key
″server write key″, client_random + server_random)[0..15]

This maps to the Cryptographic API's `CpaCyKeyGenTlsOpData` as follows:

这映射到加密 API 的 CpaCyKeyGenTlsOpData，如下所示:

```
TLS User Defined Derivation:
```

TLS 用户定义的派生:

```
tlsOp = CPA_CY_KEY_TLS_OP_USER_DEFINED
```

tlsOp = CPA _ CY _ KEY _ TLS _ OP _用户定义

```
secret = server_write_key
```

secret =服务器写密钥

```
seed = client_random + server_random
userLabel = "server write key"
```

seed = client _ random+server _ random user label = ″服务器写入密钥″

5. In the *Transport Layer Security (TLS) Protocol Version 1.0 standard*, RFC 2246, refer to Table 1., Section 6.3 iv_block is described as:

```
iv_block = PRF("", "IV block", client_random + server_random)
```

6. 在传输层安全(TLS)协议版本 1.0 标准 RFC 2246 中，请参考 Table 1

iv_block = PRF(″″, ″IV block″, client_random + server_random)[0..15]

This maps to the Cryptographic API's `CpaCyKeyGenTlsOpData` as follows:

这映射到加密 API 的 CpaCyKeyGenTlsOpData，'如下所示:

```
TLS User Defined Derivation:
```

TLS 用户定义的派生:

```
    tlsOp = CPA_CY_KEY_TLS_OP_USER_DEFINED
```

tlsOp = CPA _ CY _ KEY _ TLS _ OP _用户定义

```
    secret = NULL
```

秘密=空

```
    seed = client_random + server_random
    userLabel = "IV block"
```

seed = client _ random+server _ random 用户标签=
″IV block ″

Memory for the user label must be physically contiguous memory allocated by the user. This memory must be available to the API for the duration of the operation.

用户标签的内存必须是用户分配的物理上连续的内存。在操作期间，API 必须可以使用这个内存。

3.2.23 Session Update for Chained Cipher and Hash Operation

3.2.24 链式密码和哈希运算的会话更新

This example demonstrates the usage of the session update together with data plane symmetric API to perform a “chained” cipher and hash operation. It performs a `KASUMI F9` hash operation on the sample text and then encrypts the sample text using the `KASUMI F8` algorithm. After the operation is complete, the cipher and authentication keys are updated inside the session and the operation is performed again with different keys.

此示例演示了会话更新与数据平面对称 API 一起使用来执行“链式”密码和散列操作。它对样本文本执行 KASUMI F9 哈希运算，然后使用 KASUMI F8 算法对样本文本进行加密。操作完成后，在会话中更新密码和身份验证密钥，并使用不同的密钥再次执行操作。

Note: This example is simplified to demonstrate the basics of how to use the API and how to build the structures required. This example does not demonstrate the optimal way to use the API to get maximum performance for a particular implementation. Refer to Table 1 for implementation specific documentation and performance sample code for a guide on how to use the API for best performance.
注意:这个例子被简化以演示如何使用 API 以及如何构建所需结构的基础知识。这个例子没有展示使用 API 来获得特定实现的最佳性能的最佳方式。涉及 Table 1

These samples are located in `/sym/update_sample`
这些示例位于/sym/update_sample 中

The following are the details of the steps performed in the sample:

以下是示例中执行的步骤的详细信息:

- Cryptographic service instances are discovered and started in the same way and using the same API as the traditional symmetric use cases described in Listing 4, Listing 5 and Listing 6.
- 发现和启动加密服务实例的方式和使用的 API 与中描述的传统对称用例相同 Listing 4Listing 5 Listing 6
- Next register a callback function for the cryptographic instance.
- 接下来为加密实例注册一个回调函数。

  The function is called back in the context of the polling function when an asynchronous operation has completed. This function can perform whatever processing is appropriate to the application. Note this differs from the traditional symmetric API where the callback function is registered for the session.

  当异步操作完成时，在轮询函数的上下文中回调该函数。这个函数可以执行任何适合应用程序的处理。注意，这不同于传统的对称 API，在传统的对称 API 中，回调函数是为会话注册的。

Listing 47. Register Callback Function
清单 47。注册回调函数

```
status = cpaCySymDpRegCbFunc(cyInstHandle, symDpCallback);
```

```
status = cpaCySymDpRegCbFunc(cyInstHandle，symDpCallback);
```

3.2.12.1 Create and Initialize a Session:

3.2.12.1 创建并初始化会话:C

Listing 48. Create and Initialize Data Plane Session

清单 48。创建并初始化数据平面会话

```
    sessionSetupData.sessionPriority = CPA_CY_PRIORITY_HIGH;
    sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
    sessionSetupData.algChainOrder =
        CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;
    sessionSetupData.cipherSetupData.cipherAlgorithm =
        CPA_CY_SYM_CIPHER_KASUMI_F8;
    sessionSetupData.cipherSetupData.pCipherKey = pCipherKey;
    sessionSetupData.cipherSetupData.cipherKeyLenInBytes =
    cipherKeyLen; sessionSetupData.cipherSetupData.cipherDirection =
        CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
    sessionSetupData.hashSetupData.hashAlgorithm =
CPA_CY_SYM_HASH_KASUMI_F9;
    sessionSetupData.hashSetupData.hashMode =
    CPA_CY_SYM_HASH_MODE_AUTH;
```

```
    sessionSetupData.sessionPriority = CPA_CY_PRIORITY_HIGH;
    sessionSetupData.symOperation = CPA_CY_SYM_OP_ALGORITHM_CHAINING;
    sessionSetupData.algChainOrder =
        CPA_CY_SYM_ALG_CHAIN_ORDER_HASH_THEN_CIPHER;
    sessionSetupData.cipherSetupData.cipherAlgorithm =
        CPA_CY_SYM_CIPHER_KASUMI_F8;
    sessionSetupData.cipherSetupData.pCipherKey = pCipherKey;
    sessionSetupData.cipherSetupData.cipherKeyLenInBytes = cipherKeyLen;
    sessionSetupData.cipherSetupData.cipherDirection =
        CPA_CY_SYM_CIPHER_DIRECTION_ENCRYPT;
    sessionSetupData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_KASUMI_F9;
    sessionSetupData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_AUTH;
    sessionSetupData.hashSetupData.digestResultLenInBytes = DIGEST_LENGTH;
```

```
sessionSetupData.hashSetupData.authModeSetupData.authKey = authKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
    authKeyLen;
sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes =
    sizeof(additionalAuthData);
sessionSetupData.digestIsAppended =
CPA_TRUE; sessionSetupData.verifyDigest =
CPA_FALSE;

/* Determine size of session context to allocate */
PRINT_DBG("cpaCySymDpSessionCtxGetSize\n");
status = cpaCySymDpSessionCtxGetSize(
    cyInstHandle, &sessionSetupData, &sessionCtxSize);

if (CPA_STATUS_SUCCESS == status)
{
    /* Allocate session context */
    status = PHYS_CONTIG_ALLOC(sessionCtx, sessionCtxSize);
}

if (CPA_STATUS_SUCCESS == status)
{
    /* Initialize the session */
    PRINT_DBG("cpaCySymDpInitSession\n");
    status = cpaCySymDpInitSession(cyInstHandle, &sessionSetupData,
                                   *sessionCtx);
```

```
sessionSetupData.hashSetupData.authModeSetupData.authKey = authKey;
sessionSetupData.hashSetupData.authModeSetupData.authKeyLenInBytes =
    authKeyLen; sessionSetupData.hashSetupData.authModeSetupData.aadLenInBytes =
    sizeof(additionalAuthData);
sessionSetupData.digestIsAppended = CPA_TRUE;
sessionSetupData.verifyDigest = CPA_FALSE;

/* Determine size of session context to allocate */
PRINT_DBG("cpaCySymDpSessionCtxGetSize\n");
status = cpaCySymDpSessionCtxGetSize(
    cyInstHandle, &sessionSetupData, &sessionCtxSize);

if (CPA_STATUS_SUCCESS == status)
{
    /* Allocate session context */
    status = PHYS_CONTIG_ALLOC(sessionCtx, sessionCtxSize);
}

if (CPA_STATUS_SUCCESS == status)
{
    /* Initialize the session */
    PRINT_DBG("cpaCySymDpInitSession\n");
    status = cpaCySymDpInitSession(cyInstHandle, &sessionSetupData,
                                   *sessionCtx);
}
```

In this example, data is stored in flat buffers (as opposed to scatter gather lists). The operational data in this case is:

在本例中，数据存储在平面缓冲区中(与分散收集列表相反)。这种情况下的操作数据是:

Listing 49. Data Plane Operational Data
清单 49。数据平面操作数据

```
pOpData->thisPhys = sampleVirtToPhys(pOpData);
pOpData->instanceHandle = cyInstHandle; pOpData-
>sessionCtx = sessionCtx;
pOpData->pCallbackTag = (void *)0; pOpData-
>cryptoStartSrcOffsetInBytes = 0;
pOpData->messageLenToCipherInBytes = srcLen;
pOpData->hashStartSrcOffsetInBytes = 0; pOpData-
>messageLenToHashInBytes = srcLen;
pOpData->digestResult = sampleVirtToPhys(pSrcBuffer) + srcLen; pOpData-
>iv = sampleVirtToPhys(pIvBuffer);
pOpData->pIv = pIvBuffer;
pOpData->ivLenInBytes = ivLen;
pOpData->additionalAuthData =
sampleVirtToPhys(pAdditionalAuthData);
```

```
pOpData->thisPhys =
sampleVirtToPhys(pOpData); pOpData-
>instanceHandle = cyInstHandle; pOpData-
>sessionCtx = sessionCtx;
pOpData->pCallbackTag = (void *)0;
pOpData->cryptoStartSrcOffsetInBytes =
0;
pOpData->messageLenToCipherInBytes =
srcLen; pOpData->hashStartSrcOffsetInBytes
= 0; pOpData->messageLenToHashInBytes =
srcLen;
pOpData->digestResult = sampleVirtToPhys(pSrcBuffer) +
srcLen; pOpData->iv = sampleVirtToPhys(pIvBuffer);
pOpData->pIv = pIvBuffer;
```

```
pOpData->srcBuffer =
sampleVirtToPhys(pSrcBuffer); pOpData-
>srcBufferLen = bufferSize;
```

```
pOpData->srcBuffer = sampleVirtToPhys(pSrcBuffer);
pOpData->srcBufferLen = bufferSize;
pOpData->dstBuffer = sampleVirtToPhys(pDstBuffer);
```

This request is then enqueued on the instance.

然后，该请求在实例上排队。

Listing 50. Data Plane Enqueue
清单 50。数据平面排队

```
status = cpaCySymDpEnqueueOp(pOpData, CPA_FALSE);
```

```
status = cpaCySymDpEnqueueOp(pOpData, CPA _ FALSE);
```

Other requests can now be enqueued before submitting all the requests to be processed. This allows the cost of submitting a request (which can be expensive, in terms of cycles, for some hardware-based implementations) to be amortized over all enqueued requests on the instance. Once sufficient requests have been enqueued they are all submitted for processing.

现在，在提交所有要处理的请求之前，可以将其他请求排队。这使得提交请求的成本(对于一些基于硬件的实现来说，在周期方面可能很昂贵)可以分摊到实例上所有排队的请求中。一旦有足够多的请求进入队列，它们都会被提交进行处理。

Listing 51. Data Plane Perform
清单 51。数据平面执行

```
status = cpaCySymDpPerformOpNow(cyInstHandle);
```

```
status = cpaCySymDpPerformOpNow(cyInstHandle);
```

- An alternative to calling the `cpaCySymDpPerformOpNow` function is to set `performOpNow` to `CPA_TRUE` when calling the enqueue functions (`cpaCySymDpEnqueueOp` or `cpaCySymDpEnqueueOpBatch`). This is illustrated in the data compression data plane example.
- 调用 cpaCySymDpPerformOpNow 函数的另一种方法是在调用入队函数(cpaCySymDpEnqueueOp 或 cpaCySymDpEnqueueOpBatch)时将 PerformOpNow 设置为 CPA_TRUE。数据压缩数据平面示例说明了这一点。
- After submitting a number of requests and possibly doing other work (e.g. enqueuing and submitting more requests) the application can poll for responses which will invoke the callback function registered with the instance. Refer to Table 1 for implementation specific documentation for information on the implementations polling functions.
- 在提交大量请求并可能执行其他工作(例如，排队并提交更多请求)后，应用程序可以轮询响应，这些响应将调用向实例注册的回调函数。涉及 Table 1
- After the operation is complete cipher key and authentication key are updated in the existing session via session update API:
- 操作完成后，通过会话更新 API 在现有会话中更新加密密钥和身份验证密钥:

Listing 52. Session Update
清单 52。会话更新

```
sessionUpdateData.flags = CPA_CY_SYM_SESUPD_CIPHER_KEY;
sessionUpdateData.flags |= CPA_CY_SYM_SESUPD_AUTH_KEY;
sessionUpdateData.pCipherKey = pCipherKey;
sessionUpdateData.authKey = authKey;
[illegible]
```

sessionUpdateData.flags = CPA_CY_SYM_SESUPD_CIPHER_KEY;
sessionUpdateData.flags |= CPA_CY_SYM_SESUPD_AUTH_KEY;
sessionUpdateData.pCipherKey = pCipherKey; sessionUpdateData.authKey =
authKey;

- With the keys changed, the chained cipher and hash operation is performed again, just as described above.
- 随着密钥的改变，再次执行链式密码和散列操作，就像上面描述的那样。
- Once all requests associated with a session have been completed, the session can be removed.
- 一旦完成了与会话相关联的所有请求，就可以删除该会话。

3.2.25 HKDF Use Case
3.2.26 HKDF 用例

This section contains sample code that demonstrates the usage of the symmetric API, specifically using this API to perform hash-based message authentication code key derivation function (HKDF) operations. It performs HKDF Extract and Expand, and Extract and Expand Label operation without and with sublabels (KEY and IV).

本节包含演示对称 API 用法的示例代码，特别是使用该 API 执行基于哈希的消息身份验证代码密钥派生函数(HKDF)操作。它执行 HKDF 提取和扩展，以及提取和扩展标签操作，不使用和使用子标签(键和 IV)。

The simplified code example below is simplified and demonstrates how to use the API and build the structures required. This example does not demonstrate the optimal way to use the API to get maximum performance for implementation.
下面的简化代码示例经过了简化，演示了如何使用 API 和构建所需的结构。这个例子并没有展示使用 API 来获得最佳实现性能的最佳方式。

This sample is located in the directory:
该示例位于以下目录中：

`/quickassist/lookaside/access_layer/src/sample_code/functional/sym/`

/quickassist/lookaside/access_layer/src/sample_code/functional/sym/hkdf_s ample

3.2.13.1 Instance Configuration and Memory Allocation

3.2.13.2 实例配置和内存分配

Cryptographic service instances are discovered and started in the same way and using the same API as the traditional symmetric use cases.

加密服务实例的发现和启动方式与传统对称用例相同，使用相同的 API。

1. If the instance is polled, start the polling thread.
2. 如果实例被轮询，则启动轮询线程。

   Polling is done in an implementation-dependent manner.

   轮询是以依赖于实现的方式完成的。
3. Allocate memory for HKDF operation data:
4. 为 HKDF 操作数据分配内存：

Listing 53. HKDF Operation Data – Memory Allocation
清单 53。HKDF 操作数据-内存分配

```
pOpData = qaeMemAllocNUMA(sizeof(CpaCyKeyGenHKDFOpData),
                          instanceInfo2.nodeAffinity,
```

pOpData = qaeMemAllocNUMA(sizeof(CpaCyKeyGenHKDFOpData), instanceInfo2.nodeAffinity, BYTE_ALIGNMENT_64);

   This structure must be allocated with USDM to be pinned in physical memory.

   该结构必须分配有 USDM，才能固定在物理内存中。
5. Allocate memory for HKDF output data. Output data is `CpaFlatBuffer` type:
6. 为 HKDF 输出数据分配内存。输出数据是 CpaFlatBuffer 类型：

3.2.13.3 HKDF Extract Expand Operation

3.2.13.4 HKDF 提取扩展操作

To perform an Extract Expand operation, go to the `CpaCyKeyGenHKDFOpData` structure, and set `hkdfKeyOp` to `CPA_CY_HKDF_KEY_EXTRACT_EXPAND`.

若要执行提取扩展操作，请转到 CpaCyKeyGenHKDFOpData 结构，并将 hkdfKeyOp 设置为 CPA_CY_HKDF_KEY_EXTRACT_EXPAND。

Note: Provide the lengths `seedLen`, `secretLen`, and `infoLen` and copy all data into the seed, secret, and info tables.
注意：提供长度 seedLen、secretLen 和 infoLen，并将所有数据复制到 seed、secret 和 info 表中。

Listing 54. HKDF Extract Expand Operation

清单 54。HKDF 提取扩展操作

```
pOpData->hkdfKeyOp = CPA_CY_HKDF_KEY_EXTRACT_EXPAND;
pOpData->seedLen = sizeof(ikm);
memcpy(pOpData->seed, ikm, pOpData->seedLen);
pOpData->secretLen = sizeof(slt);
memcpy(pOpData->secret, slt, pOpData->secretLen);
pOpData->infoLen = sizeof(inf);
memcpy(pOpData->info, inf, pOpData->infoLen);
```

```
pOpData->hkdfKeyOp = CPA_CY_HKDF_KEY_EXTRACT_EXPAND;
pOpData->seedLen = sizeof(ikm); memcpy(pOpData->seed, ikm, pOpData->seedLen);
pOpData->secretLen = sizeof(slt);
memcpy(pOpData->secret, slt, pOpData->secretLen);
pOpData->infoLen = sizeof(inf); memcpy(pOpData->info, inf, pOpData->infoLen);
```

3.2.13.5 HKDF Extract Expand Label Operation

3.2.13.6 HKDF 提取扩展标签操作

To perform an Extract, Expand Label operation:

要执行提取，请展开标签操作:

1. Go to the CpaCyKeyGenHKDFOpData structure and set hkdfKeyOp to
2. 转到 CpaCyKeyGenHKDFOpData 结构并将 hkdfKeyOp 设置为
CPA_CY_HKDF_KEY_EXTRACT_EXPAND_LABEL.
注册会计师_ CY _ HKDF _关键_提取_扩展_标签。

3. Provide the lengths `seedLen`, `secretLen`, and `infoLen` and copy all data into the
4. 提供长度 seedLen、secretLen 和 infoLen，并将所有数据复制到
   `seed`, `secret`, and `info` tables.
   种子、机密和信息表。
5. Set the number of labels in the `numLabels` field.
6. 在“数字标签”字段中设置标签的数量。
7. Set `label[0].labelLen`.
8. 设置标签[0].labelLen。
9. Copy label data into the `label[0].label` table.
10. 将标签数据复制到标签[0]中。标签表。
11. Finally, set the `label[0].sublabelFlag` field to `0x00` to disable generating sublabels.
12. 最后，设置标签[0]。sublabelFlag 字段设置为 0x00，以禁止生成子标签。

Listing 55. HKDF Extract Expand Label Operation
清单 55。HKDF 提取扩展标签操作

```
pOpData->hkdfKeyOp = CPA_CY_HKDF_KEY_EXTRACT_EXPAND_LABEL;
pOpData->seedLen = sizeof(seed_label);
memcpy(pOpData->seed, seed_label, sizeof(seed_label));

pOpData->secretLen = sizeof(secret_label);
memcpy(pOpData->secret, secret_label, sizeof(secret_label));

pOpData->numLabels = 1;
memcpy(pOpData->label[0].label, label, sizeof(label));
pOpData->label[0].labelLen = sizeof(label); pOpData->label[0].sublabelFlag = 0x00;
```

```
pOpData->hkdfKeyOp = CPA_CY_HKDF_KEY_EXTRACT_EXPAND_LABEL;
pOpData->seedLen = sizeof(seed_label);
memcpy(pOpData->seed, seed_label, sizeof(seed_label));
pOpData->secretLen = sizeof(secret_label);
memcpy(pOpData->secret, secret_label, sizeof(secret_label));
pOpData->numLabels = 1;
memcpy(pOpData->label[0].label, label, sizeof(label));
pOpData->label[0].labelLen = sizeof(label);
pOpData->label[0].sublabelFlag = 0x00;
```

3.2.13.7 HKDF Extract Expand Label and Sublabels operation

3.2.13.8 HKDF 提取展开标签和子标签操作

To perform an Extract, Expand Label and Sublabels operation:

要执行提取、展开标签和子标签操作：

1. Go to the `CpaCyKeyGenHKDFOpData` structure and set `hkdfKeyOp` to `CPA_CY_HKDF_KEY_EXTRACT_EXPAND_LABEL`.
2. 转到 CpaCyKeyGenHKDFOpData 结构并将 hkdfKeyOp 设置为
   注册会计师_ CY_HKDF_关键_提取_扩展_标签。
3. Provide the lengths `seedLen`, `secretLen`, and `infoLen` and copy all data into the seed, secret, and info tables.
4. 提供长度 seedLen、secretLen 和 infoLen，并将所有数据复制到 seed、secret 和 info 表中。
5. Set the number of labels in the `numLabels` field.
6. 在“数字标签”字段中设置标签的数量。
7. Set `label[0].labelLen`.
8. 设置标签[0].labelLen。
9. Copy label data into the `label[0].label` table.
10. 将标签数据复制到标签[0]中。标签表。
11. Set the `label[0].sublabelFlag` and `label[0].sublabelFlag` field as shown below to generate Key and IV sublabels.
12. 设置标签[0]。子标签和标签[0]。sublabelFlag 字段如下所示，用于生成 Key 和 IV 子标签。

Listing 56. HKDF Extract Expand Label and Sublabels Operation
清单 56。HKDF 提取展开标签和子标签操作

```
pOpData->hkdfKeyOp = CPA_CY_HKDF_KEY_EXTRACT_EXPAND_LABEL;
pOpData->seedLen = sizeof(seed_label);
memcpy(pOpData->seed, seed_label, sizeof(seed_label));

pOpData->secretLen = sizeof(secret_label);
memcpy(pOpData->secret, secret_label, sizeof(secret_label));
```

```
pOpData->hkdfKeyOp = CPA_CY_HKDF_KEY_EXTRACT_EXPAND_LABEL;
pOpData->seedLen = sizeof(seed_label);
memcpy(pOpData->seed, seed_label, sizeof(seed_label));
pOpData->secretLen = sizeof(secret_label);
memcpy(pOpData->secret, secret_label, sizeof(secret_label));
```

```
pOpData->label[0].labelLen = sizeof(label);
pOpData->label[0].sublabelFlag =
```

```
pOpData->label[0].labelLen = sizeof(label);
pOpData->label[0].sublabelFlag = CPA_CY_HKDF_SUBLABEL_KEY; pOpData-
```

3.2.27 Perform HKDF operation

3.2.28 执行 HKDF 操作

The crypto instance must be specified in the `instanceHandle` to execute the HKDF operation. When the operation is performed asynchronously, the callback function and callback tag should be set in the `pKeyGenCb` and `pCallbackTag` arguments.

要执行 HKDF 操作，必须在 instanceHandle 中指定 crypto 实例。异步执行操作时，回调函数和回调标记应在 pKeyGenCb 和 pCallbackTag 参数中设置。

Operational data is provided in `pKeyGenTlsOpData,` and `CpaCyKeyHKDFCipherSuite` must be chosen. The output is passed to the `CpaFlatBuffer`. All generated values are arranged one after the other in a single buffer. Depending on what operations are performed, the buffer length should be adjusted.

操作数据在 pKeyGenTlsOpData 中提供，必须选择 CpaCyKeyHKDFCipherSuite。输出被传递到 CpaFlatBuffer。所有生成的值在单个缓冲区中一个接一个地排列。根据所执行的操作，应该调整缓冲区长度。

Listing 57. HKDF Operation
清单 57。HKDF 行动

```
cpaCyKeyGenTls3(cyInstHandle, /* Instance handle */
                hkdfSampleCallback, /* Callback function
                */ (void *)&complete, /* Callback tag */
                pOpData, /* HKDF operational data */
                CPA_CY_HKDF_TLS_AES_128_GCM_SHA256, /* HKDF cipher
                suite
```

```
cpaCyKeyGenTls3(cyInstHandle, /* Instance handle */
                hkdfSampleCallback, /* Callback function */ (void
                *)&complete, /* Callback tag */ pOpData, /* HKDF
                operational data */
                CPA_CY_HKDF_TLS_AES_128_GCM_SHA256, /* HKDF cipher suite
```

6.5 Using the Diffie-Hellman API

6.6 使用 Diffie-Hellman API

This example demonstrates the usage of the Diffie-Hellman API.

These samples are located in `/asym/diffie_hellman_sample`

这个例子演示了 Diffie-Hellman API 的用法。这些示例位

于/asym/diffie_hellman_sample 中

The following steps are carried out:
执行以下步骤:

- The example uses the API asynchronously; therefore, you must define a Diffie-Hellman callback function per the API prototype.
- 该示例异步使用 API 因此，您必须为每个 API 原型定义一个 Diffie- Hellman 回调函数。
- Instance, discovery, and start-up are made in a way similar to that defined for the symmetric examples above.
- 实例、发现和启动的方式类似于上面对称示例中定义的方式。

- The function `sampleDhPerformOp` is called, which does the following:
- 调用函数 sampleDhPerformOp，它执行以下操作:
    - Allocate memory for the operation and populate data for the appropriate DH phase 1 operation data structure to generate the public value. The fields to be allocated and populated are the prime P, the base G, and the private value X. Space must also be allocated for the output, which is the public value (PV).
    - 为操作分配内存，并为适当的DH phase 1操作数据结构填充数据，以生成公共值。要分配和填充的字段是质数P、基数G和私有值x。还必须为公共值(PV)的输出分配空间。

Listing 58. Allocate Memory and Populate Operational Data
清单58。分配内存并填充操作数据

```
status = OS_MALLOC(&pCpaDhOpDataP1, sizeof(CpaCyDhPhase1KeyGenOpData));

/*
 * Allocate input buffers for phase 1 and copy data. Input to DH
 * phase 1 includes the prime (primeP), the base g (baseG) and
 * a random private value (privateValueX).
 */
```

```
status = OS_MALLOC(&pCpaDhOpDataP1, sizeof(CpaCyDhPhase1KeyGenOpData));

/*
 * Allocate input buffers for phase 1 and copy data. Input to DH
 * phase 1 includes the prime (primeP), the base g (baseG) and
 * a random private value (privateValueX).
 */
```

```
    memset(pCpaDhOpDataP1, 0, sizeof(CpaCyDhPhase1KeyGenOpData));

    pCpaDhOpDataP1->primeP.dataLenInBytes = sizeof(primeP_768);
    status =
        PHYS_CONTIG_ALLOC(&pCpaDhOpDataP1->primeP.pData,
sizeof(primeP_768));

    if (NULL != pCpaDhOpDataP1->primeP.pData) {
        memcpy(pCpaDhOpDataP1->primeP.pData,
        primeP_768,
sizeof(primeP_768));
    }
}

if (CPA_STATUS_SUCCESS == status) {
    pCpaDhOpDataP1->baseG.dataLenInBytes = sizeof(baseG1);
    status = PHYS_CONTIG_ALLOC(&pCpaDhOpDataP1-
    >baseG.pData,
sizeof(baseG1));

    if (NULL != pCpaDhOpDataP1->baseG.pData) {
        memcpy(pCpaDhOpDataP1->baseG.pData, baseG1, sizeof(baseG1));
    }
}

if (CPA_STATUS_SUCCESS == status) {
    pCpaDhOpDataP1->privateValueX.dataLenInBytes =
    sizeof(privateValueX); status = PHYS_CONTIG_ALLOC(&pCpaDhOpDataP1-
    >privateValueX.pData,
                               sizeof(privateValueX));

    if (NULL != pCpaDhOpDataP1->privateValueX.pData)
        { memcpy(pCpaDhOpDataP1-
    memset(pCpaDhOpDataP1, 0, sizeof(CpaCyDhPhase1KeyGenOpData));

    pCpaDhOpDataP1->primeP.dataLenInBytes = sizeof(primeP_768); status =
        PHYS_CONTIG_ALLOC(&pCpaDhOpDataP1->primeP.pData, sizeof(primeP_768));

    if (NULL != pCpaDhOpDataP1->primeP.pData) {
        memcpy(pCpaDhOpDataP1->primeP.pData, primeP_768,
sizeof(primeP_768));
    }
}

if (CPA_STATUS_SUCCESS == status) {
    pCpaDhOpDataP1->baseG.dataLenInBytes = sizeof(baseG1); status =
    PHYS_CONTIG_ALLOC(&pCpaDhOpDataP1->baseG.pData,
sizeof(baseG1));

    if (NULL != pCpaDhOpDataP1->baseG.pData) {
        memcpy(pCpaDhOpDataP1->baseG.pData, baseG1, sizeof(baseG1));
    }
}

if (CPA_STATUS_SUCCESS == status) {
    pCpaDhOpDataP1->privateValueX.dataLenInBytes = sizeof(privateValueX); status =
    PHYS_CONTIG_ALLOC(&pCpaDhOpDataP1->privateValueX.pData,
                                       sizeof(privateValueX));

    if (NULL != pCpaDhOpDataP1->privateValueX.pData) {
        memcpy(pCpaDhOpDataP1->privateValueX.pData,
                privateValueX, sizeof(privateValueX));
    }
}
```

Invoke the phase 1 operation, which performs the modular exponentiation such that PV = (baseG ^ privateValueX) mod `primeP`.

调用阶段 1 操作，该操作执行模幂运算，使得 PV =(baseg ˆ privatevaluex)mod primap。

Note: In the case of phase 1, the operation is invoked synchronously, hence the NULL pointer for the callback function.
注意:在阶段 1 的情况下，操作是同步调用的，因此回调函数的指针为空。

Listing 59. Perform Phase 1 Operation
清单 59。执行阶段 1 操作

```
status =
status = cpaCyDhKeyGenPhase2Secret(
    cpaCyDhKeyGenPhase2Secret( cy
    cyInstHandle,
    InstHandle,
    (const CpaCyGenFlatBufCbFunc)asymCallback, /* CB function*/
    pCpaDhOpDataP2,         /* structure containing p, the public value &
x*/
    pOctetStringSecretKey); /* private key (output of the function)*/
```

```
    pCpaDhOpDataP2,         /* structure containing p, the public value &
x*/
    pOctetStringSecretKey); /* private key (output of the function)*/
```

In a real-world implementation of a key exchange protocol, the public value generated above would now be shared with another party, B. This example uses this public value to go on and invoke the second phase operation. First allocate memory for the secret value, setup the operational data for the phase 2 operation, and then perform that operation. This operation is invoked asynchronously, taking the callback function defined earlier as a parameter:

在密钥交换协议的实际实现中，上面生成的公共值现在将与另一方 b 共享。该示例使用该公共值继续并调用第二阶段操作。首先为秘密值分配内存，为阶段 2 操作设置操作数据，然后执行该操作。这个操作是异步调用的，将前面定义的回调函数作为参数:

Listing 60. Perform Phase 2 Operation

清单 60。执行第 2 阶段操作

```
            status =
                cpaCyDhKeyGenPhase2Secret( cy
                InstHandle,
                (const CpaCyGenFlatBufCbFunc)asymCallback, /* CB function*/
                pCallbackTagPh2,         /* pointer to the complete variable*/
                pCpaDhOpDataP2,          /* structure containing p, the public value
```

```
            status = cpaCyDhKeyGenPhase2Secret(
                cyInstHandle,
                (const CpaCyGenFlatBufCbFunc)asymCallback, /* CB function*/
                pCallbackTagPh2,         /* pointer to the complete variable*/
                pCpaDhOpDataP2,          /* structure containing p, the public value &
x*/
```

Finally, clean up by freeing up memory, stopping the instance, etc.

最后，通过释放内存、停止实例等方式进行清理。

3.3.1 Prime Number Testing

3.3.2 素数测试

This example demonstrates the usage of the prime number testing API.

These samples are located in `/asym/prime_sample`

这个例子演示了素数测试 API 的用法。这些示例位于/asym/prime_sample 中

The following steps are carried out:
执行以下步骤:

- The API is used asynchronously: therefore, a callback function is defined as per the API prototype.
- API 是异步使用的:因此，回调函数是按照 API 原型定义的。
- Instance, discovery, and start-up is made in a way similar to that defined for the symmetric examples above.
- 实例、发现和启动的方式类似于上面对称示例中定义的方式。
- The function `primePerformOp` is called, which does the following:
- 调用 primePerformOp 函数，该函数执行以下操作:
  - Allocate memory for the operation
  - 为操作分配内存
  - Populate data for the appropriate input fields and perform the operation. The fields populated include the following:

- 为适当的输入字段填充数据并执行操作。填充的字段包括以下内容:
    - Prime Candidate
    - 首要候选人
    - Whether to perform the greatest common divisor (GCD) test
    - 是否执行最大公约数(GCD)测试
    - Whether to perform the Fermat test
    - 是否执行费马测试
    - Number of Miller-Rabin rounds
    - 米勒-拉宾回合数
    - Whether to perform Lucas test
    - 是否进行卢卡斯测试

Listing 61. Setup Operational Data and Test Prime

清单 61。设置运行数据并测试灌注

```
pPrimeTestOpData->primeCandidate.pData = pPrime;
pPrimeTestOpData->primeCandidate.dataLenInBytes =
sizeof(samplePrimeP_768);
pPrimeTestOpData->performGcdTest = CPA_TRUE;
```

```
pPrimeTestOpData->primeCandidate.pData = pPrime;
pPrimeTestOpData->primeCandidate.dataLenInBytes =
sizeof(samplePrimeP_768);
pPrimeTestOpData->performGcdTest = CPA_TRUE;
```

```
pPrimeTestOpData->numMillerRabinRounds = NB_MR_ROUNDS;
pPrimeTestOpData->millerRabinRandomInput.pData = pMR;
pPrimeTestOpData->millerRabinRandomInput.dataLenInBytes =
sizeof(MR); pPrimeTestOpData->performLucasTest = CPA_TRUE;

status = cpaCyPrimeTest(
                 cyInstHandle,
                 (const CpaCyPrimeTestCbFunc)primeCallback, /* CB
function */
           (void *)&complete, /* callback tag */
           pPrimeTestOpData, /* operation data */
           &testPassed); /* return value: true if the number is
           probably
```

```
pPrimeTestOpData->numMillerRabinRounds = NB_MR_ROUNDS; pPrimeTestOpData-
>millerRabinRandomInput.pData = pMR; pPrimeTestOpData-
>millerRabinRandomInput.dataLenInBytes = sizeof(MR); pPrimeTestOpData-
>performLucasTest = CPA_TRUE;

status = cpaCyPrimeTest(
                 cyInstHandle,
                 (const CpaCyPrimeTestCbFunc)primeCallback, /* CB
function */
           (void *)&complete, /* callback tag */
           pPrimeTestOpData, /* operation data */
           &testPassed); /* return value: true if the number is probably
```

Finally, statistics are queried and the service stopped.

最后，查询统计数据并停止服务。

6.7 Using the SM2 API
6.8 使用 SM2 API

This example demonstrates the usage of the SM2 API.

The following steps are carried out:

这个例子演示了 SM2 API 的用法。执行以下步骤:

- The example contains synchronous and asynchronous API. For latter one, you must define a proper callback function per the API prototype for different SM2 operations.
- 该示例包含同步和异步 API。对于后一种情况，您必须根据 API 原型为不同的 SM2 操作定义适当的回调函数。

- Instance, discovery, and start-up are made in a way similar to that defined for the symmetric examples above.
- 实例、发现和启动的方式类似于上面对称示例中定义的方式。
- This sample involves sign/verify, encryption/decryption, key exchange, point multiplication/verify.
- 该示例涉及签名/验证、加密/解密、密钥交换、点乘/验证。

3.4.1 SM2 Digital Signature Generation and Verification
3.4.2 SM2 数字签名生成和验证

This operation is to sign a given message and output its digital signature (r,s) then verify (r,s).

该操作是对给定的消息进行签名，并输出其数字签名(r，s)，然后验证(r，s)。

- The function `sampleEcsm2SignPerformOp` provisions parts of example implementation, which does the following:
- 函数 sampleEcsm2SignPerformOp 提供了示例实现的一部分，它执行以下操作:
  - Allocate memory and populate data for input buffer which includes scalar multiplier k, digest of the message e and private key d.
  - 为输入缓冲区分配内存和填充数据，输入缓冲区包括标量乘数 k、消息摘要 e 和私钥 d。
  - Allocate memory for output buffer which includes signature r and s.
  - 为包含签名 r 和 s 的输出缓冲区分配内存。
  - Call function `cpaCyEcsm2Sign` for sign operation.
  - 调用函数 cpaCyEcsm2Sign 进行符号操作。

```
status = OS_MALLOC(&pCpaEcsm2SignOpData, sizeof(CpaCyEcsm2SignOpData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2SignOpData->k.pData, sizeof(k));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2SignOpData->e.pData, sizeof(e));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2SignOpData->d.pData, sizeof(d));
status = PHYS_CONTIG_ALLOC(&pR->pData, GFP_SM2_SIZE_IN_BYTE);
```

status = OS_MALLOC(&pCpaEcsm2SignOpData, sizeof(CpaCyEcsm2SignOpData)); status = PHYS_CONTIG_ALLOC(&pCpaEcsm2SignOpData->k.pData, sizeof(k)); status = PHYS_CONTIG_ALLOC(&pCpaEcsm2SignOpData->e.pData, sizeof(e)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2SignOpData->d.pData, sizeof(d));
status = PHYS_CONTIG_ALLOC(&pR->pData, GFP_SM2_SIZE_IN_BYTE); status =

```
status = cpaCyEcsm2Sign(
            cyInstHandle,
            (const CpaCyEcsm2SignCbFunc)asymSignCallback, /* CB
function*/
            pCallbackTag,                                 /* Opaque
            user
data */
            pCpaEcsm2SignOpData, /* Structure containing k, d and e */
            &signStatus, /* signStatus indicates if the result is valid
*/
            pR           /* Signature r (function output) */
```

```
status = cpaCyEcsm2Sign(
            cyInstHandle,
            (const CpaCyEcsm2SignCbFunc)asymSignCallback, /* CB
function*/
            pCallbackTag,                                 /* Opaque user
data */
            pCpaEcsm2SignOpData, /* Structure containing k, d and e */ &signStatus,
            /* signStatus indicates if the result is valid
*/
            pR,          /* Signature r (function output) */
```

- The function `sampleEcsm2VerifyPerformOp` provisions parts of example implementation, which does the following:
- 函数 sampleEcsm2VerifyPerformOp 提供了示例实现的一部分，它执行以下操作：
    - Allocate memory and populate data for input buffer which includes digest of the message e, signature r and s, x coordinate of public key and y coordinate of public key.
    - 为输入缓冲区分配内存和填充数据，包括消息 e 的摘要、签名 r 和 s、公钥的 x 坐标和公钥的 y 坐标。
    - Call function `cpaCyEcsm2Verify` for signature verification operation.
    - 调用函数 cpaCyEcsm2Verify 进行签名验证操作。

```
status =
OS_MALLOC(&pCpaEcsm2VerifyOpData,
sizeof(CpaCyEcsm2VerifyOpData));

status = PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->e.pData, sizeof(e));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->r.pData, sizeof(r));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->s.pData, sizeof(s));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->xP.pData,
sizeof(xP));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->yP.pData,
sizeof(yPA));

status = cpaCyEcsm2Verify(
            cyInstHandle,
            (const CpaCyEcsm2VerifyCbFunc)asymVerifyCallback, /* CB
function*/
            pCallbackTag              /* Opaque user data */
```

```
status = OS_MALLOC(&pCpaEcsm2VerifyOpData,
sizeof(CpaCyEcsm2VerifyOpData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->e.pData, sizeof(e)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->r.pData, sizeof(r)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->s.pData, sizeof(s)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->xP.pData, sizeof(xP));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2VerifyOpData->yP.pData, sizeof(yPA));
status = cpaCyEcsm2Verify(
            cyInstHandle,
            (const CpaCyEcsm2VerifyCbFunc)asymVerifyCallback, /* CB
function*/
            pCallbackTag,             /* Opaque user data */
            pCpaEcsm2VerifyOpData, /* Verify request data*/
            &verifyStatus);           /* verify status */
```

3.4.3 SM2 Public Key Encryption

3.4.4 SM2 公钥加密

This operation is to encrypt a given message then decrypt the cipher and compare to the given message.

该操作是加密给定的消息，然后解密密码并与给定的消息进行比较。

- The function `sampleEcsm2EncPerformOp` provisions parts of example implementation, which does the following:

- 函数 sampleEcsm2EncPerformOp 提供了示例实现的一部分，它执行以下操作:
  - Allocate memory and populate data for input buffer which includes scalar multiplier k, x coordinate of public key xP and y coordinate of public key yP.
  - 为输入缓冲区分配内存和填充数据，输入缓冲区包括标量乘数 k、公钥 xP 的 x 坐标和公钥 yP 的 y 坐标。
  - Allocate memory for output buffer which includes x coordinate of [k]G x1, y coordinate of [k]G y1, x coordinate of [k]Pb x2 and y coordinate of [k]Pb y2.
  - 为包含[k]G x1 的 x 坐标、[k]G y1 的 y 坐标、[k]Pb x2 的 x 坐标和[k]Pb y2 的 y 坐标的输出缓冲区分配内存。

- Call function `cpaCyEcsm2Encrypt` for encryption operation.
- 调用函数 cpaCyEcsm2Encrypt 进行加密操作。

```
status = OS_MALLOC(&pCpaEcsm2EncOpData,
sizeof(CpaCyEcsm2EncryptOpData)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOpData->k.pData, sizeof(k)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOpData->xP.pData, sizeof(xP)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOpData->yP.pData, sizeof(yP));
```

- The function `sampleEcsm2DecPerformOp` provisions parts of example implementation, which does the following:
- 函数 sampleEcsm2DecPerformOp 提供了示例实现的一部分，它执行以下操作：
  - Allocate memory and populate data for input buffer which includes private key d, x coordinate of [k]G x1 and y coordinate of [k]G y1.
  - 为输入缓冲区分配内存并填充数据，输入缓冲区包括私钥 d、[k]G x1 的 x 坐标和[k]G y1 的 y 坐标。
  - Allocate memory for output buffer which includes x coordinate of [k]Pb x2 and y coordinate of [k]Pb y2.
  - 为输出缓冲区分配内存，该缓冲区包括[k]Pb x2 的 x 坐标和[k]Pb y2 的 y 坐标。
  - Call function `cpaCyEcsm2Decrypt` for decryption operation.
  - 调用函数 cpaCyEcsm2Decrypt 进行解密操作。
  - Call function `sm3` and `hashCheck` to check correctness of decryption.
  - 调用函数 sm3 和 hashCheck 来检查解密的正确性。

```
status = OS_MALLOC(&pCpaEcsm2EncOutputData,
                          sizeof(CpaCyEcsm2EncryptOutputData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOutputData->x1.pData,
                          GFP_SM2_SIZE_IN_BYTE);
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOutputData->y1.pData,
                          GFP_SM2_SIZE_IN_BYTE);
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOutputData->x2.pData,
                          GFP_SM2_SIZE_IN_BYTE);
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOutputData->y2.pData,
                          GFP_SM2_SIZE_IN_BYTE);

status = OS_MALLOC(&pCpaEcsm2DecOpData,
sizeof(CpaCyEcsm2DecryptOpData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2DecOpData->d.pData,
sizeof(d)); status = PHYS_CONTIG_ALLOC(&pCpaEcsm2DecOpData-
>x1.pData,
                          GFP_SM2_SIZE_IN_BYTE);
     status = PHYS_CONTIG_ALLOC(&pCpaEcsm2DecOpData-
                          >y1.pData,
                          GFP_SM2_SIZE_IN_BYTE);
status = cpaCyEcsm2Encrypt(
        cyInstHandle,
        (const CpaCyGenFlatBufCbFunc)asymCallback, /* CB
function*/
        pCallbackTag,                    /* Opaque
        user
data */
status = OS_MALLOC(&pCpaEcsm2DecOutputData,
                          sizeof(CpaCyEcsm2DecryptOutputData));

status = PHYS_CONTIG_ALLOC(&pCpaEcsm2DecOutputData->x2.pData,
```

```
status = OS_MALLOC(&pCpaEcsm2EncOpData, sizeof(CpaCyEcsm2EncryptOpData)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOpData->k.pData, sizeof(k)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOpData->xP.pData, sizeof(xP)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2EncOpData->yP.pData, sizeof(yP));
status = OS_MALLOC(&pCpaEcsm2EncOutputData,
    sizeof(CpaCyEcsm2EncryptOutputData)); status =
status = OS_MALLOC(&pCpaEcsm2DecOpData, sizeof(CpaCyEcsm2DecryptOpData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2DecOpData->d.pData, sizeof(d)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2DecOpData->x1.pData,
                          GFP_SM2_SIZE_IN_BYTE);
     status = PHYS_CONTIG_ALLOC(&pCpaEcsm2DecOpData->y1.pData,
                          GFP_SM2_SIZE_IN_BYTE);
status = cpaCyEcsm2Encrypt(
        cyInstHandle,
        (const CpaCyGenFlatBufCbFunc)asymCallback, /* CB
function*/
        pCallbackTag,                    /* Opaque user
data */
        pCpaEcsm2EncOpData, /* Encryption request data */
status = OS_MALLOC(&pCpaEcsm2DecOutputData,
sizeof(CpaCyEcsm2DecryptOutputData)); status =
PHYS_CONTIG_ALLOC(&pCpaEcsm2DecOutputData->x2.pData,
                          GFP_SM2_SIZE_IN_BYTE);
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2DecOutputData->y2.pData,
status = cpaCyEcsm2Decrypt(
            cyInstHandle,
            (const CpaCyGenFlatBufCbFunc)asymDecCallback, /* CB
function*/
            pCallbackTag,                                 /* Opaque
            user
data */
            pCpaEcsm2DecOpData, /* Decryption request data */
            pCpaEcsm2DecOutputData /* Decryption response data */);


sm3(pDecOutputData, MESSAGE_LEN + 2 * GFP_SM2_SIZE_IN_BYTE,
                   pHashBuffer);
```

```
status = cpaCyEcsm2Decrypt(
            cyInstHandle,
            (const CpaCyGenFlatBufCbFunc)asymDecCallback, /* CB
function*/
            pCallbackTag,                                 /* Opaque user
data */
            pCpaEcsm2DecOpData, /* Decryption request data */
            pCpaEcsm2DecOutputData /* Decryption response data */);


sm3(pDecOutputData, MESSAGE_LEN + 2 * GFP_SM2_SIZE_IN_BYTE,
                   pHashBuffer); hashCheck(pC3Buffer,
```

3.4.5 SM2 Key Exchange

3.4.6 SM2 密钥交换

This operation is to exchange key between A side and B side, and check if the shared keys are the same.

该操作是在 A 侧和 B 侧之间交换密钥，并检查共享密钥是否相同。

- The function `sampleEcsm2KeyExPerformOp` provisions parts of example implementation, which does the following:
- 函数 sampleEcsm2KeyExPerformOp 提供了示例实现的一部分，它执行以下操作:
  - Allocate phase 1 input buffer which includes scalar multiplier r for A side and B side separately.
  - 分别为 A 侧和 B 侧分配包含标量乘法器 r 的阶段 1 输入缓冲器。
  - Allocate phase 1 output buffer which includes x coordinate of a point on the curve x and y coordinate of a point on the curve y for for A side and B side separately.
  - 为 A 侧和 B 侧分别分配包含曲线 x 上一点的 x 坐标和曲线 y 上一点的 y 坐标的阶段 1 输出缓冲器。
  - Call function `cpaCyEcsm2KeyExPhase1` for A side and B side separately.

- 分别为 A 侧和 B 侧调用函数 cpaCyEcsm2KeyExPhase1。
- Allocate phase 2 input buffer which includes scalar multiplier r, private key d, x coordinate of a point on the curve from other side x1, x coordinate of a point on the curve from phase 1 x2, y coordinate of a point on the curve from phase 1 y2, x coordinate of public key from other side xP and y coordinate of public key from other side yP for A side and B side separately.
- 为 A 侧和 B 侧分别分配阶段 2 输入缓冲器，其包括标量乘数 r、私钥 d、来自另一侧 x1 的曲线上的点的 x 坐标、来自阶段 1 x2 的曲线上的点的 x 坐标、来自阶段 1 y2 的曲线上的点的 y 坐标、来自另一侧 xP 的公钥的 x 坐标和来自另一侧 yP 的公钥的 y 坐标。
- Allocate phase 2 input buffer which includes x coordinate of a point on the curve x and y coordinate of a point on the curve y for A side and B side separately.
- 为 A 侧和 B 侧分别分配阶段 2 输入缓冲区，其中包括曲线 x 上某点的 x 坐标和曲线 y 上某点的 y 坐标。
- Call function `cpaCyEcsm2KeyExPhase2` for A side and B side separately.
- 分别为 A 侧和 B 侧调用函数 cpaCyEcsm2KeyExPhase2。

```
status = OS_MALLOC(&pCpaEcsm2KeyExPhase1AOpData,
                   sizeof(CpaCyEcsm2KeyExPhase1OpData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase1AOpData->r.pData,
                           sizeof(rA));
```

status = OS_MALLOC(&pCpaEcsm2KeyExPhase1AOpData, sizeof(CpaCyEcsm2KeyExPhase1OpData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase1AOpData->r.pData,
sizeof(rA));

```
status = OS_MALLOC(&pCpaEcsm2KeyExPhase1BOpData,
                   sizeof(CpaCyEcsm2KeyExPhase1OpData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase1BOpData->r.pData,
                           sizeof(rB));
```

status = OS_MALLOC(&pCpaEcsm2KeyExPhase1BOpData, sizeof(CpaCyEcsm2KeyExPhase1OpData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase1BOpData->r.pData,
sizeof(rB));

```
status = OS_MALLOC(&pCpaEcsm2KeyExPhase1AOutputData,
```

status = OS_MALLOC(&pCpaEcsm2KeyExPhase1AOutputData,

```
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase1AOutputData->x.pData,
status = PHYS _重叠_分配(& pcpaecsm 2 keyexphase 1 output data-> x . pdata
                                           GFP_SM2_SIZE_IN_BYTE);
                                           GFP _ SM2 _大小_字节);
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase1AOutputData->y.pData,
status = PHYS _重叠_分配(& pcpaecsm 2 keyexphase 1 output data-> y . pdata
                                           GFP_SM2_SIZE_IN_BYTE);
                                           GFP _ SM2 _大小_字节);


status = OS_MALLOC(&pCpaEcsm2KeyExPhase1BOutputData,
status = OS _ MALLOC(& pcpaecsm 2 keyexphase 1 boutputdata,
                           sizeof(CpaCyEcsm2KeyExOutputData));
                           sizeof(cpacyecsm 2 keyexoutputdata));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase1BOutputData->x.pData,
status = PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 1 boutputdata-> x . pdata
                                           GFP_SM2_SIZE_IN_BYTE);
                                           GFP _ SM2 _大小_字节);
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase1BOutputData->y.pData,
status = PHYS _重叠_分配(& pcpaecsm 2 keyexphase 1 boutputdata-> y . pdata
                                           GFP_SM2_SIZE_IN_BYTE);
                                           GFP _ SM2 _大小_字节);


status = cpaCyEcsm2KeyExPhase1(
status = cpaCyEcsm2KeyExPhase1(
            cyInstHandle,
            cyInstHandle,
            (const CpaCyGenFlatBufCbFunc)
                asymKeyExPhase1Callback, /* CB function*/
            (const CpaCyGenFlatBufCbFunc)asymkeyexphase 1 callback, /* CB
                函数*/
            pCallbackTag,                /* Opaque user data */
            pCpaEcsm2KeyExPhase1AOpData, /* Key exchange p1 request
            data
            pCallbackTag, /* Opaque 用户数据*/ pCpaEcsm2KeyExPhase1AOpData, /*密钥交
            换 p1 请求数据

*/
*/


data
*/);
data */);
```

```
pCpaEcsm2KeyExPhase    exchange p1 response
1AOutputData /* Key    pcpaecsm 2 keyexphase 1 output data/*密钥交换 p1 响应


          status = cpaCyEcsm2KeyExPhase1(
          status = cpaCyEcsm2KeyExPhase1(
                       cyInstHandle,
                       cyInstHandle,
                       (const CpaCyGenFlatBufCbFunc)
                           asymKeyExPhase1Callback, /* CB function*/
                       (const CpaCyGenFlatBufCbFunc)asymkeyexphase 1 callback, /* CB
                           函数*/
                       pCallbackTag,                /* Opaque user data */
                       pCpaEcsm2KeyExPhase1BOpData, /* Key exchange p1 request
                       data
                       pCallbackTag, /* Opaque 用户数据*/ pCpaEcsm2KeyExPhase1BOpData, /*密钥交
                       换 p1 请求数据

          */
          */


          data*/);
          data
          */);
```

```
pCpaEcsm2KeyExPha        Key exchange p1 request
se1BOutputData /*        pcpaecsm 2 keyexphase 1 boutputdata/*密钥交换 p1 请求

         status = OS_MALLOC(&pCpaEcsm2KeyExPhase2AOpData,
         status = OS _ MALLOC(& pcpaecsm 2 keyexphase 2 aopdata,
                                   sizeof(CpaCyEcsm2KeyExPhase2OpData));
         status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2AOpData->r.pData,
                                   sizeof(cpacyecsm 2 keyexphase 2 opdata)); status =
         PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 aopdata-> r . pdata
                                           sizeof(rA));
                                           sizeof(rA));
         status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2AOpData->d.pData,
         status = PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 aopdata-> d . pdata
                                           sizeof(dA));
                                           sizeof(dA));
         status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2AOpData->x1.pData,
         pCpaEcsm2KeyExPhase2AOpData->x1.dataLenInBytes);
         status = PHYS _重叠_分配(& pcpaecsm 2 keyexphase 2 aopdata-> x1 .
         pdata, pcpaecsm 2 keyexphase 2 aopdata-> x1 . dataleninbytes);
         status = PHYS_CONTIG_ALLOC(
         状态= PHYS _重叠_分配(
                 &pCpaEcsm2KeyExPhase2AOpData->x2.pData,
                 pCpaEcsm2KeyExPhase1BOutputData->x.dataLenInBytes);
                 & pcpaecsm 2 keyexphase 2 aopdata-> x2 . pdata, pcpaecsm 2
                 keyexphase 1 boutputdata-> x . dataleninbytes);
         status = PHYS_CONTIG_ALLOC(
         状态= PHYS _重叠_分配(
                 &pCpaEcsm2KeyExPhase2AOpData->y2.pData,
                 pCpaEcsm2KeyExPhase1BOutputData->y.dataLenInBytes);
                 & pcpaecsm 2 keyexphase 2 aopdata-> y2 . pdata, pcpaecsm 2
                 keyexphase 1 boutputdata-> y . dataleninbytes);
```

```
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2AOpData->xP.pData,
status = PHYS _重叠_分配(& pcpaecsm 2 keyexphase 2 aopdata-> XP . pdata
                                          sizeof(xPB));
                                          sizeof(xPB));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2AOpData->yP.pData,
status = PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 aopdata-> yp . pdata
                                          sizeof(yPB));
                                          sizeof(yPB));


status = OS_MALLOC(&pCpaEcsm2KeyExPhase2BOpData,
status = OS _ MALLOC(& pcpaecsm 2 keyexphase 2 bop data,
                                 sizeof(CpaCyEcsm2KeyExPhase2OpData));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2BOpData->r.pData,

                                 sizeof(cpacyecsm 2 keyexphase 2 opdata)); status =
PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 bop data-> r . pdata
                                          sizeof(rB));
                                          sizeof(rB));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2BOpData->d.pData,
status = PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 bop data-> d . pdata
                                          sizeof(dB));
                                          sizeof(dB));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2BOpData->x1.pData,
status = PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 bop data-> x1 . pdata,
                                    pCpaEcsm2KeyExPhase2BOpData -
                                    pCpaEcsm2KeyExPhase2BOpData -
>x1.dataLenInBytes);
> x1 . dataleninbytes);
status = PHYS_CONTIG_ALLOC(
         &pCpaEcsm2KeyExPhase2BOpData->x2.pData,
status = PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 bop
         data-> x2 . pdata,
              pCpaEcsm2KeyExPhase1AOutputData->x.dataLenInBytes);
status = PHYS_CONTIG_ALLOC(
              pcpaecsm 2 keyexphase 1a output data-> x . dataleninbytes); 状态=
PHYS _重叠_分配(
              &pCpaEcsm2KeyExPhase2BOpData->y2.pData,
              pCpaEcsm2KeyExPhase1AOutputData->y.dataLenInBytes);
              & pcpaecsm 2 keyexphase 2 bop data-> y2 . pdata, pcpaecsm 2
              keyexphase 1 output data-> y . dataleninbytes);
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2BOpData->xP.pData,
status = PHYS _重叠_分配(& pcpaecsm 2 keyexphase 2 bop data-> XP . pdata
                                          sizeof(xPA));
                                          sizeof(xPA));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2BOpData->yP.pData,
status = PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 bop data-> yp . pdata
                                   sizeof(yPA));
                                   sizeof(yPA));

status = OS_MALLOC(&pCpaEcsm2KeyExPhase2AOutputData,
status = OS _ MALLOC(& pcpaecsm 2 keyexphase 2 output data,
                           sizeof(CpaCyEcsm2KeyExOutputData));
                           sizeof(cpacyecsm 2 keyexoutputdata));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2AOutputData->x.pData,
status = PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 output data-> x . pdata
                                   GFP_SM2_SIZE_IN_BYTE);
                                   GFP _ SM2 _大小_字节);
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2AOutputData->y.pData,
status = PHYS _重叠_分配(& pcpaecsm 2 keyexphase 2 output data-> y . pdata
                                   GFP_SM2_SIZE_IN_BYTE);
                                   GFP _ SM2 _大小_字节);


status = OS_MALLOC(&pCpaEcsm2KeyExPhase2BOutputData,
status = OS _ MALLOC(& pcpaecsm 2 keyexphase 2 boutputdata,
                           sizeof(CpaCyEcsm2KeyExOutputData));
                           sizeof(cpacyecsm 2 keyexoutputdata));
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2BOutputData->x.pData,
status = PHYS _重叠群_分配(& pcpaecsm 2 keyexphase 2 boutputdata-> x . pdata
                                   GFP_SM2_SIZE_IN_BYTE);
                                   GFP _ SM2 _大小_字节);
status = PHYS_CONTIG_ALLOC(&pCpaEcsm2KeyExPhase2BOutputData->y.pData,
status = PHYS _重叠_分配(& pcpaecsm 2 keyexphase 2 boutputdata-> y . pdata
                                   GFP_SM2_SIZE_IN_BYTE);
                                   GFP _ SM2 _大小_字节);


status = cpaCyEcsm2KeyExPhase2(
status = cpaCyEcsm2KeyExPhase2(
            cyInstHandle,
            cyInstHandle,
            (const CpaCyGenFlatBufCbFunc)asymKeyExPhase2Callback,
            pCallbackTag,                /* Opaque user data; */
            pCpaEcsm2KeyExPhase2AOpData, /* Key exchange p2 request
            data,
            (const CpaCyGenFlatBufCbFunc)asymkeyexphase 2 callback, pCallbackTag, /*
            Opaque 用户数据; */ pCpaEcsm2KeyExPhase2AOpData, /*密钥交换 p2 请求数据,
                                         containing
                                         包含
r,d,x1,x2,y2,xp,yp*/
r、d、x1、x2、y2、xp、yp*/
```

```
            pCpaEcsm2KeyExPhase2AOutputData /* Key exchange p2 response
*/);

status = cpaCyEcsm2KeyExPhase2(
            cyInstHandle,
            (const CpaCyGenFlatBufCbFunc)asymKeyExPhase2Callback,
            pCallbackTag,                /* Opaque user data */
            pCpaEcsm2KeyExPhase2BOpData, /* Key exchange p2 request
            data
                                            containing
r,d,x1,y1,x2,y2,xp,yp*/
```

```
            pCpaEcsm2KeyExPhase2AOutputData /* Key exchange p2 response
*/);

status = cpaCyEcsm2KeyExPhase2(
            cyInstHandle,
            (const CpaCyGenFlatBufCbFunc)asymKeyExPhase2Callback, pCallbackTag,
                                         /* Opaque user data */
            pCpaEcsm2KeyExPhase2BOpData, /* Key exchange p2 request data
                                            containing
r,d,x1,y1,x2,y2,xp,yp*/
            pCpaEcsm2KeyExPhase2BOutputData /* Key exchange p2 response
```

3.4.7 SM2 Elliptic Curve Point

3.4.8 SM2 椭圆曲线点

This operation is to calculate a point on the curve according to a given random number and verify if the point (x,y) on the curve or not.

这个操作是根据给定的随机数计算曲线上的一个点，并验证该点(x，y)是否在曲线上。

- The function `sampleEcsm2PointMultiply` provisions parts of example implementation, which does the following:
- 函数 sampleEcsm2PointMultiply 提供了示例实现的一部分，它执行以下操作:
  - Allocate memory and populate date for input buffer which includes scalar multiplier k, x coordinate of a point on the curve x and y coordinate of a point on the curve y.
  - 为输入缓冲区分配内存和填充数据，输入缓冲区包括标量乘数 k、曲线 x 上某点的 x 坐标和曲线 y 上某点的 y 坐标。
  - Allocate memory for output buffer which includes x coordinate of the resulting point multiplication pXk and y coordinate of the resulting point multiplication pYk.
  - 为输出缓冲区分配内存，该缓冲区包括结果点乘 pXk 的 x 坐标和结果点乘 pYk 的 y 坐标。
  - Call function `cpaCyEcsm2PointMultiply` for point multiply operation.
  - 调用函数 cpaCyEcsm2PointMultiply 进行点乘运算。

```
status = OS_MALLOC(&opData,
sizeof(CpaCyEcsm2PointMultiplyOpData)); status =
PHYS_CONTIG_ALLOC(&opData->k.pData, sizeof(k));
status = PHYS_CONTIG_ALLOC(&opData->x.pData, sizeof(xP));
status = PHYS_CONTIG_ALLOC(&opData->y.pData, sizeof(yP));

status = OS_MALLOC(&pXk,
sizeof(CpaFlatBuffer)); status =
```

```
status = OS_MALLOC(&opData, sizeof(CpaCyEcsm2PointMultiplyOpData)); status =
PHYS_CONTIG_ALLOC(&opData->k.pData, sizeof(k));
status = PHYS_CONTIG_ALLOC(&opData->x.pData, sizeof(xP)); status =
PHYS_CONTIG_ALLOC(&opData->y.pData, sizeof(yP));

status = OS_MALLOC(&pXk, sizeof(CpaFlatBuffer)); status
= OS_MALLOC(&pYk, sizeof(CpaFlatBuffer));
```

```
                );
);
```

- The function `sampleEcsm2GeneratortMultiply` provisions parts of example implementation, which does the following:
- 函数 sampleEcsm2GeneratortMultiply 提供了示例实现的一部分，它执行以下操作：
  - Allocate memory and populate date for input buffer which includes scalar multiplier k.
  - 为包含标量乘数 k 的输入缓冲区分配内存和填充数据。
  - Allocate memory for output buffer which includes x coordinate of the resulting point multiplication pXk and y coordinate of the resulting point multiplication pYk.
  - 为输出缓冲区分配内存，该缓冲区包括结果点乘 pXk 的 x 坐标和结果点乘 pYk 的 y 坐标。
  - Call function `cpaCyEcsm2GeneratorMultiply` for generator multiply operation.
  - 调用函数 cpaCyEcsm2GeneratorMultiply 进行生成器乘法运算。
- The function `sampleEcsm2PointVerify` provisions parts of example implementation, which does the following:
- 函数 sampleEcsm2PointVerify 提供了示例实现的一部分，它执行以下操作：
  - Allocate memory and populate date for input buffer which includes x coordinate of a point on the curve x and y coordinate of a point on the curve y.
  - 为输入缓冲区分配内存和填充数据，输入缓冲区包括曲线 x 上某点的 x 坐标和曲线 y 上某点的 y 坐标。
  - Call function `cpaCyEcsm2PointVerify` for EC point verification.
  - 调用函数 cpaCyEcsm2PointVerify 进行 EC 点验证。

```
status = OS_MALLOC(&opData, sizeof(CpaCyEcsm2GeneratorMultiplyOpData));
status = PHYS_CONTIG_ALLOC(&opData->k.pData, sizeof(k));
status = OS_MALLOC(&pXk,
sizeof(CpaFlatBuffer)); status =
OS_MALLOC(&pYk, sizeof(CpaFlatBuffer));
status = PHYS_CONTIG_ALLOC(&pXk->pData, GFP_SM2_SIZE_IN_BYTE);
status = PHYS_CONTIG_ALLOC(&pYk->pData, GFP_SM2_SIZE_IN_BYTE);
status = OS_MALLOC(&opData, sizeof(CpaCyEcsm2PointVerifyOpData));
status = PHYS_CONTIG_ALLOC(&opData->x.pData, sizeof(x));
status = PHYS_CONTIG_ALLOC(&opData->y.pData, sizeof(y));
status = cpaCyEcsm2GeneratorMultiply(cyInstHandle,
        (const CpaCyEcPointMultiplyCbFunc)asymGeneratorMultCallback,/* CB function*/
        pCallbackTag,                   /* Opaque user data */
        opData,   /* Generator multiplication request data */
        &multiplyStatus,
        pXk,  /* Generator multiplication response data */
        pYk); /* Generator multiplication response data */
status = cpaCyEcsm2PointVerify(cyInstHandle, (const CpaCyEcPointVerifyCbFunc)asymPointVerifyCallback, /* CB
function */
pCallbackTag,                   /* Opaque user data; */
        opData,                         /* Point multiplication request
data */
        &multiplyStatus,
```

```
status = OS_MALLOC(&opData, sizeof(CpaCyEcsm2GeneratorMultiplyOpData)); status =
PHYS_CONTIG_ALLOC(&opData->k.pData, sizeof(k));
status = OS_MALLOC(&pXk, sizeof(CpaFlatBuffer)); status
= OS_MALLOC(&pYk, sizeof(CpaFlatBuffer));
status = PHYS_CONTIG_ALLOC(&pXk->pData, GFP_SM2_SIZE_IN_BYTE); status =
PHYS_CONTIG_ALLOC(&pYk->pData, GFP_SM2_SIZE_IN_BYTE);
status = OS_MALLOC(&opData, sizeof(CpaCyEcsm2PointVerifyOpData)); status =
PHYS_CONTIG_ALLOC(&opData->x.pData, sizeof(x));
status = PHYS_CONTIG_ALLOC(&opData->y.pData, sizeof(y));
status = cpaCyEcsm2GeneratorMultiply(
        cyInstHandle,
        (const CpaCyEcPointMultiplyCbFunc)asymGeneratorMultCallback,
        /* CB function*/
        pCallbackTag,                   /* Opaque user data */
        opData, /* Generator multiplication request data */
        &multiplyStatus,
        pXk,  /* Generator multiplication response data */
        pYk); /* Generator multiplication response data */
status = cpaCyEcsm2PointVerify(cyInstHandle, (const CpaCyEcPointVerifyCbFunc)
        asymPointVerifyCallback, /* CB
pCallbackTag,                 /* Opaque user data */
        opData,                        /* Point multiplication request
data */
        &multiplyStatus,
        pXk, /* Point multiplication response data */ pYk /*
        Point multiplication response data */
```

```
                                        pCallbackTag, /* Opaque user data
                                        */ opData,    /* Point verify
                                        request
data */
                                        &verifyStatus);

            status =
cpaCyEcsm2GeneratorMultiply( cyInst
```

```
                                        pCallbackTag, /* Opaque user data */
                                        opData,         /* Point verify request
data */
                                        &verifyStatus);

  status = cpaCyEcsm2GeneratorMultiply(
            cyInstHandle,
              (const CpaCyEcPointMultiplyCbFunc)
```

Finally, statistics are queried and the service stopped.

最后，查询统计数据并停止服务。

§

7 Intel® QuickAssist Technology Data Compression API

8 英特尔快速辅助技术数据压缩 API

This chapter describes the sample code for the Intel® QuickAssist Technology Data Compression API, beginning with an API overview, and followed by descriptions of various scenarios to illustrate the usage of the API.

本章介绍英特尔 QuickAssist 技术数据压缩 API 的示例代码，首先是 API 概述，然后是说明 API 用法的各种场景描述。

Note: This document does not cover data integrity concepts. Refer to Table 1 in the Programmer's Guide, Compress and Verify (CnV) Related APIs for your product for important information on data integrity concepts, including the Compress-and-Verify feature.
注意:本文档不包含数据完整性概念。涉及 Table 1

8.1 Overview

8.2 概观

The Intel® QuickAssist Technology Data Compression API can be categorized into three broad areas as follows:

英特尔 QuickAssist 技术数据压缩 API 可分为以下三大类:

- Common: This includes functionality for the initialization and shutdown of the service.
- Common:这包括服务的初始化和关闭功能。
- Instance Management: A given implementation of the API can present multiple instances of the compression service, each representing a logical or virtual "device". Request order is guaranteed within a given instance of the service.
- 实例管理:给定的 API 实现可以呈现压缩服务的多个实例，每个实例代表一个逻辑或虚拟“设备”。在给定的服务实例中保证请求顺序。
- Transformation:
- 转型:
  - Compression functionality
  - 压缩功能
  - Decompression functionality
  - 解压缩功能

These areas of functionality are defined in `cpa_dc.h` and `cpa_dc_dp.h`.
这些功能领域在 cpa_dc.h 和 cpa_dc_dp.h 中定义。

The Intel® QAT Data Compression API uses the "base" API (cpa), which defines base data types used across all services of the Intel® QAT Technology API.

英特尔 QAT 数据压缩 API 使用“基本”API (cpa)，它定义了英特尔 QAT 技术 API 的所有服务所使用的基本数据类型。

4.1.1 Session
4.1.2 会议

Similar to the symmetric cryptography API, the data compression API has the concept of a session. In the case of the compression API, a session is an object that describes the compression parameters to be applied across several requests. These requests might submit buffers within a single file, or buffers associated with a particular data stream or flow. A session object is described by the following:

与对称加密 API 类似，数据压缩 API 也有会话的概念。在压缩 API 的情况下，会话是一个对象，它描述了要跨几个请求应用的压缩参数。这些请求可能提交单个文件中的缓冲区，或者与特定数据流或流相关联的缓冲区。会话对象描述如下:

- **The compression level**: Lower levels provide faster compression and the cost of compression ratio, whereas higher levels provide a better compression ratio as the cost of performance.
- 压缩级别:较低的级别提供更快的压缩速度和压缩比成本，而较高的级别提供更好的压缩比和性能成本。
- **The compression algorithm:** to use (e.g. deflate) and what type of Huffman trees to use (static or dynamic).
- 压缩算法:使用(例如 deflate)和使用什么类型的霍夫曼树(静态或动态)。
- **The session direction**: If all requests on this session are compression requests, then the direction can be set to compress (and similarly, for decompress). A combined direction is also available if both compression and decompression requests are called using this session.
- 会话方向:如果这个会话上的所有请求都是压缩请求，那么可以将方向设置为压缩(同样，对于解压缩也是如此)。如果使用此会话调用压缩和解压缩请求，则组合方向也是可用的。

- **The session state**: a session can be described as stateful or stateless. Stateful sessions maintain history and state between calls to the API, and stateless sessions do not.
- 会话状态:会话可以描述为有状态的或无状态的。有状态会话维护 API 调用之间的历史和状态，而无状态会话不这样做。
  - Stateless compression does not require history data from a previous compression/decompression request to be restored before submitting the request. Stateless sessions are used when the output data is known to be constrained in size. An overflow condition (when the output data is about to exceed the output buffer) is treated as an error condition in the decompression direction. In the compression direction, the Application can keep submitting data from where the overflow was registered in the input stream. The Data Plane API treats overflow as an error. In this case, the overflow is treated as an error rather than an exception. The client application is required to resubmit the job in its entirety with a larger output buffer. Requests are treated independently; state and history are not saved and restored between calls.
  - 无状态压缩不要求在提交请求之前恢复来自先前压缩/解压缩请求的历史数据。当已知输出数据的大小受限时，使用无状态会话。溢出情况(当输出数据即将超出输出缓冲区时)被视为解压缩方向的错误情况。在压缩方向，应用程序可以从输入流中注册溢出的地方继续提交数据。数据平面 API 将溢出视为错误。在这种情况下，溢出被视为错误而不是异常。客户端应用程序需要使用更大的输出缓冲区重新提交整个作业。请求被独立处理；在两次调用之间不会保存和恢复状态和历史记录。

Note: When using a stateless session, it is possible to feed a seed checksum to the `cpaDcCompressData()` or the `cpaDcDecompressData()` API when the `CPA_DC_FLUSH_FULL` flush flag is used. The user application is responsible for maintaining the checksum across requests. This feature is also known as Stateful Lite.
注意:使用无状态会话时，如果使用了 CPA_DC_FLUSH_FULL flush 标志，则可以向 cpaDcCompressData() 或 cpaDcDecompressData() API 提供种子校验和。用户应用程序负责维护跨请求的校验和。这个特性也被称为有状态 Lite。

  - Stateful sessions are required when the data to be decompressed is larger than the buffers being used. This is a standard mode of operation for applications such as GZIP, where the size of the uncompressed data is not known before execution, and therefore the destination buffer may not be large enough to hold the resultant output. Requests to stateful sessions are not treated independently, and state and history can be saved and restored between calls. The amount of history and state carried between calls depends on the compression level. For stateful decompression, only one outstanding request may be in-flight at any one time for that session.
  - 当要解压缩的数据大于正在使用的缓冲区时，需要有状态会话。这是 GZIP 等应用程序的标准操作模式，在这些应用程序中，未压缩数据的大小在执行前是未知的，因此目标缓冲区可能不够大，无法容纳最终输出。对有状态会话的请求不是独立处理的，状态和历史可以在调用之间保存和恢复。调用之间传递的历史和状态的数量取决于压缩级别。对于有状态解压缩，对于该会话，任何时候都只能有一个未完成的请求。

8.3 Sample – Stateful Data Compression
8.4 示例 – 有状态数据压缩

This example demonstrates the usage of the synchronous API, specifically using this API to perform a compression operation. It compresses a file via a stateful session using the deflate compress algorithm with static Huffman trees and using GZIP style headers and footers.

这个例子演示了同步 API 的用法，特别是使用这个 API 来执行压缩操作。它使用 deflate 压缩算法和静态霍夫曼树，并使用 GZIP 风格的页眉和页脚，通过有状态会话压缩文件。

These samples are located in `/dc/stateful_sample`
这些示例位于/dc/stateful_sample 中

Note: Stateful data compression is not available in Intel® QAT v1.8 and later releases. However, stateful decompression is available in Intel® QAT v1.8 and later releases.
注意:英特尔 QAT v1.8 版及更高版本中不提供有状态数据压缩。但是，英特尔 QAT v1.8 版和更高版本中提供了状态解压缩。

4.2.1 Session Establishment
4.2.2 会话建立

This is the main entry point for the sample compression code. It demonstrates the sequence of calls to be made to the API to create a session, perform one or more compress operations, and then tear down the session. At this point, the instance has been discovered and started, and the capabilities of the instance have been queried and found to be suitable.

这是示例压缩代码的主要入口点。它演示了调用 API 来创建会话、执行一个或多个压缩操作，然后拆除会话的顺序。此时，实例已经被发现并启动，并且实例的能力已经被查询并发现是合适的。

A session is established by describing a session, determining how much session memory is required, and then invoking the session initialization function `cpaDcInitSession`.

通过描述会话、确定需要多少会话内存，然后调用会话初始化函数 cpaDcInitSession 来建立会话。

Listing 62. Create and Initialize Stateful Session

清单 62。创建和初始化有状态会话

```
sd.compLevel = CPA_DC_L4;
sd.compType = CPA_DC_DEFLATE;
sd.huffType = CPA_DC_HT_STATIC;
sd.sessDirection = CPA_DC_DIR_COMBINED;
sd.sessState = CPA_DC_STATEFUL;
#if (CPA_DC_API_VERSION_NUM_MAJOR == 1 && CPA_DC_API_VERSION_NUM_MINOR < 6)
sd.deflateWindowSize = 7;
#endif
sd.checksum = CPA_DC_CRC32;
/* Determine size of session context to allocate */
PRINT_DBG("cpaDcGetSessionSize\n");
status = cpaDcGetSessionSize(dcInstHandle, &sd, &sess_size, &ctx_size);
if (CPA_STATUS_SUCCESS == status) {
    /* Allocate session memory */
    status = PHYS_CONTIG_ALLOC(&sessionHdl, sess_size);
}
if ((CPA_STATUS_SUCCESS == status) && (ctx_size != 0)) {
    /* Allocate context bufferlist */
    status = cpaDcBufferListGetMetaSize(dcInstHandle, 1, &buffMetaSize);
    if (CPA_STATUS_SUCCESS == status) {
        status = PHYS_CONTIG_ALLOC(&pBufferMeta, buffMetaSize);
    }
    if (CPA_STATUS_SUCCESS == status) {
        status = OS_MALLOC(&pBufferCtx, bufferListMemSize);
    }
    if (CPA_STATUS_SUCCESS == status) {
        status = PHYS_CONTIG_ALLOC(&pCtxBuf, ctx_size);
    }
    if (CPA_STATUS_SUCCESS == status) {
        pFlatBuffer = (CpaFlatBuffer *)(pBufferCtx + 1);
        pBufferCtx->pBuffers = pFlatBuffer;
        pBufferCtx->numBuffers = 1;
        pBufferCtx->pPrivateMetaData = pBufferMeta;
        pFlatBuffer->dataLenInBytes =
        ctx_size; pFlatBuffer->pData =
        pCtxBuf;
    }
}
/* Initialize the Stateful session
*/ if (CPA_STATUS_SUCCESS == status)
{
    PRINT_DBG("cpaDcInitSession\n");
    status = cpaDcInitSession(dcInstHandle,
                              sessionHdl, /* session memory */
                              &sd,        /* session setup data
                              */ pBufferCtx, /* context buffer
```

```
        pFlatBuffer->dataLenInBytes = ctx_size;
        pFlatBuffer->pData = pCtxBuf;
    }
}
/* Initialize the Stateful session */ if
(CPA_STATUS_SUCCESS == status) {
    PRINT_DBG("cpaDcInitSession\n");
    status = cpaDcInitSession(dcInstHandle,
                              sessionHdl, /* session memory */ &sd,
                                          /* session setup data */
                              pBufferCtx, /* context buffer */
                              NULL); /* callback function NULL for sync
mode */
`
```

Note: Source and destination buffers must be established.

注意:必须建立源缓冲区和目的缓冲区。

Listing 63.   Stateful Compression Memory Allocation

清单 63。有状态压缩内存分配

```
numBuffers = 1; /* only using 1 buffer in this case */
/* allocate memory for bufferlist and array of flat buffers in a
contiguous
 * area and carve it up to reduce number of memory allocations
 required.
 */
bufferListMemSize
=
    sizeof(CpaBufferList) + (numBuffers * sizeof(CpaFlatBuffer));

status = cpaDcBufferListGetMetaSize(dcInstHandle, numBuffers,
&bufferMetaSize);

/* Allocate source buffer */
if (CPA_STATUS_SUCCESS == status) {
    status = PHYS_CONTIG_ALLOC(&pBufferMetaSrc, bufferMetaSize);
```

```
numBuffers = 1; /* only using 1 buffer in this case */
/* allocate memory for bufferlist and array of flat buffers in a contiguous
 * area and carve it up to reduce number of memory allocations required.
*/ bufferListMemSize
    sizeof(CpaBufferList) + (numBuffers * sizeof(CpaFlatBuffer));

status = cpaDcBufferListGetMetaSize(dcInstHandle, numBuffers, &bufferMetaSize);

/* Allocate source buffer */
if (CPA_STATUS_SUCCESS == status) {
    status = PHYS_CONTIG_ALLOC(&pBufferMetaSrc, bufferMetaSize);
}
if (CPA_STATUS_SUCCESS == status) {
}
if (CPA_STATUS_SUCCESS == status) {
    status = PHYS_CONTIG_ALLOC(&pDstBuffer, SAMPLE_BUFF_SIZE);
}

if (CPA_STATUS_SUCCESS == status) {
    /* Build source bufferList */
    pFlatBuffer = (CpaFlatBuffer *)(pBufferListSrc + 1);

    pBufferListSrc->pBuffers =
    pFlatBuffer; pBufferListSrc-
    >numBuffers = 1;
    pBufferListSrc->pPrivateMetaData = pBufferMetaSrc;

    pFlatBuffer->dataLenInBytes =
    SAMPLE_BUFF_SIZE; pFlatBuffer->pData =
    pSrcBuffer;

    /* Build destination bufferList */
    pFlatBuffer = (CpaFlatBuffer *)(pBufferListDst + 1);

    pBufferListDst->pBuffers =
    pFlatBuffer; pBufferListDst-
    >numBuffers = 1;
    pBufferListDst->pPrivateMetaData = pBufferMetaDst;
```

```
}
if (CPA_STATUS_SUCCESS == status) {
    status = PHYS_CONTIG_ALLOC(&pDstBuffer, SAMPLE_BUFF_SIZE);
}

if (CPA_STATUS_SUCCESS == status) {
    /* Build source bufferList */
    pFlatBuffer = (CpaFlatBuffer *)(pBufferListSrc + 1);

    pBufferListSrc->pBuffers = pFlatBuffer;
    pBufferListSrc->numBuffers = 1;
    pBufferListSrc->pPrivateMetaData = pBufferMetaSrc;

    pFlatBuffer->dataLenInBytes = SAMPLE_BUFF_SIZE;
    pFlatBuffer->pData = pSrcBuffer;

    /* Build destination bufferList */
    pFlatBuffer = (CpaFlatBuffer *)(pBufferListDst + 1);

    pBufferListDst->pBuffers = pFlatBuffer;
    pBufferListDst->numBuffers = 1;
    pBufferListDst->pPrivateMetaData = pBufferMetaDst;

    pFlatBuffer->dataLenInBytes = SAMPLE_BUFF_SIZE;
    pFlatBuffer->pData = pDstBuffer;
```

Listing 64. Create Header

清单 64。创建标题

At this point, the application has opened an instance, established a session, and allocated buffers. It is time to start some compress operations. To produce GZIP style compressed files, the first thing that needs to be performed is header generation.
此时，应用程序已经打开了一个实例，建立了一个会话，并分配了缓冲区。是时候开始一些压缩操作了。要生成 GZIP 风格的压缩文件，首先要做的是生成文件头。
Create a header using the following code:
使用以下代码创建一个标题：

```
/* Write RFC1952 gzip header to destination buffer */
status = cpaDcGenerateHeader(sessionHdl, pFlatBuffer, &hdr_sz);
}
if (CPA_STATUS_SUCCESS == status) {
/* write out header */
```

```
/* Write RFC1952 gzip header to destination buffer */
status = cpaDcGenerateHeader(sessionHdl, pFlatBuffer, &hdr_sz);
}
if (CPA_STATUS_SUCCESS == status) {
/* write out header */
```

`cpaDcGenerateHeader` produces a GZIP style header (compliant with *GZIP file format specification* v4.3, RFC 1952, refer to Table 1) when the session set up data is set such that `compType` is `CPA_DC_DEFLATE` and checksum is `CPA_DC_CRC32`.

cpaDcGenerateHeader 生成 GZIP 风格的报头(符合 GZIP 文件格式规范 v4.3，RFC 1952，请参考 Table 1

Note: Alternatively, a zlib style header (compliant with *ZLIB Compressed Data Format Specification*, v3.3, RFC 1950, refer to Table 1) can be produced if the session setup data is set such that `compType` is `CPA_DC_DEFLATE` and checksum is `CPA_DC_ADLER32`. This operation
注意:或者，zlib 样式的头(符合 ZLIB 压缩数据格式规范，3.3 版，RFC 1950，请参考 Table 1

demonstrates looping through a file, reading the data, invoking the data compress operation, and writing the results to the output file.
演示循环遍历文件、读取数据、调用数据压缩操作以及将结果写入输出文件。

Listing 65. Perform Stateful Compression Operation
清单 65。执行有状态压缩操作

```
pBufferListSrc->pBuffers->dataLenInBytes = 0;
while ((!feof(srcFile)) && (CPA_STATUS_SUCCESS == status)) {
    /* read from file into src buffer */
    pBufferListSrc->pBuffers->pData =
    pSrcBuffer; pBufferListSrc->pBuffers-
    >dataLenInBytes +=
        fread(pSrcBuffer + pBufferListSrc->pBuffers-
              >dataLenInBytes, 1,
              SAMPLE_BUFF_SIZE - pBufferListSrc->pBuffers-
>dataLenInBytes,
              srcFile);
    if (pBufferListSrc->pBuffers->dataLenInBytes < SAMPLE_BUFF_SIZE)
        { flush = CPA_DC_FLUSH_FINAL;
    } else {
        flush = CPA_DC_FLUSH_SYNC;
    }
    do  { PRINT_DBG("cpaDcCompressData\
        n");
        status = cpaDcCompressData(dcInstHandle,
                                   sessionHdl,
                                   pBufferListSrc, /* source buffer
                                   list
*/
                                   pBufferListDst, /* destination
                                   buffer
list */
                                   &dcResults,     /* results structure
*/
                                   flush,          /* Stateful session
                                   */ NULL);
        if (CPA_STATUS_SUCCESS != status) {
            PRINT_ERR("cpaDcCompressData failed. (status = %d)\n",
status);
            break;
        }
        /*
         * We now check the results
         */
        if ((dcResults.status != CPA_DC_OK) &&
            (dcResults.status != CPA_DC_OVERFLOW))
            {
```

```
pBufferListSrc->pBuffers->dataLenInBytes = 0;
while ((!feof(srcFile)) && (CPA_STATUS_SUCCESS == status)) {
    /* read from file into src buffer */ pBufferListSrc-
    >pBuffers->pData = pSrcBuffer; pBufferListSrc-
    >pBuffers->dataLenInBytes +=
        fread(pSrcBuffer + pBufferListSrc->pBuffers->dataLenInBytes, 1,
              SAMPLE_BUFF_SIZE - pBufferListSrc->pBuffers-
>dataLenInBytes,
              srcFile);
    if (pBufferListSrc->pBuffers->dataLenInBytes < SAMPLE_BUFF_SIZE) { flush =
        CPA_DC_FLUSH_FINAL;
    } else {
        flush = CPA_DC_FLUSH_SYNC;
    }
    do  { PRINT_DBG("cpaDcCompressData\n");
        status = cpaDcCompressData(dcInstHandle,
                                   sessionHdl,
                                   pBufferListSrc, /* source buffer list
*/
                                   pBufferListDst, /* destination buffer
list */
                                   &dcResults,     /* results structure
*/
                                   flush,          /* Stateful session */
                                   NULL);
        if (CPA_STATUS_SUCCESS != status) { PRINT_ERR("cpaDcCompressData
            failed. (status = %d)\n",
status);
            break;
        }/*
         * We now check the results
         */
        if ((dcResults.status != CPA_DC_OK) &&
            (dcResults.status != CPA_DC_OVERFLOW)) {
            PRINT_ERR("Results status not as expected (status = %d)\n",
                      dcResults.status);
            status = CPA_STATUS_FAIL;
            break;
        }

        fwrite(pDstBuffer, 1, dcResults.produced,

        dstFile); if (dcResults.consumed <=

        pBufferListSrc->pBuffers-
>dataLenInBytes) {
            pBufferListSrc->pBuffers->dataLenInBytes -=
dcResults.consumed;
            pBufferListSrc->pBuffers->pData += dcResults.consumed;
        } else {
            pBufferListSrc->pBuffers->dataLenInBytes = 0;
        }

        if (dcResults.consumed == 0 &&
            pBufferListSrc->pBuffers->dataLenInBytes > 0)
            { memcpy(pSrcBuffer,
                   pBufferListSrc->pBuffers->pData,
                   pBufferListSrc->pBuffers-
                   >dataLenInBytes);
            break;
        }
```

```
        }

        fwrite(pDstBuffer, 1, dcResults.produced, dstFile); if

        (dcResults.consumed <= pBufferListSrc->pBuffers-
>dataLenInBytes) {
            pBufferListSrc->pBuffers->dataLenInBytes -= dcResults.consumed;
            pBufferListSrc->pBuffers->pData += dcResults.consumed;
        } else {
            pBufferListSrc->pBuffers->dataLenInBytes = 0;
        }

        if (dcResults.consumed == 0 &&
            pBufferListSrc->pBuffers->dataLenInBytes > 0) {
            memcpy(pSrcBuffer,
                   pBufferListSrc->pBuffers->pData, pBufferListSrc-
                   >pBuffers->dataLenInBytes);
            break;
        }
    } while (pBufferListSrc->pBuffers->dataLenInBytes != 0 ||
             dcResults.status == CPA_DC_OVERFLOW);
}
```

Finally, a GZIP footer is generated. Similar to the call to `cpaDcGenerateHeader`, a GZIP footer (compliant with GZIP style header (compliant with *GZIP file format specification* v4.3, RFC 1952, refer to Table 1) is produced because the session setup data is set such that `compType` is `CPA_DC_DEFLATE` and checksum is `CPA_DC_CRC32`. The call to `cpaDcGenerateFooter` increments the produced field of the `CpaDcRqResults` structure by the size of the footer added. In this example, the data produced so far has already been written out to the file. As such, the produced field of the `CpaDcRqResults` structure is cleared before calling the `cpaDcGenerateFooter` function.

最后，生成一个 GZIP 页脚。类似于对 cpaDcGenerateHeader 的调用，GZIP 页脚(符合 GZIP 样式的页眉(符合 GZIP 文件格式规范 v4.3，RFC 1952，请参考 Table 1

In the event the destination buffer would be too small to accept the footer, the `cpaDcGenerateFooter()` API will return an invalid parameter error. The `cpaDcGenerateFooter()` API cannot return an overflow exception. It is application's responsibility to ensure that there is enough allocated buffer memory to append the algorithm specific footer.

如果目标缓冲区太小而无法接受页脚，cpaDcGenerateFooter() API 将返回一个无效参数错误。cpaDcGenerateFooter() API 无法返回溢出异常。应用程序负责确保有足够的分配缓冲存储器来附加算法特定的页脚。

Listing 66. Create Footer

清单 66。创建页脚

```
            dcResults.produced = 0;
        /* Write RFC1952 gzip footer to destination buffer */
            status = cpaDcGenerateFooter(sessionHdl, pFlatBuffer,
&dcResults);
}
if (CPA_STATUS_SUCCESS == status) {
    /* write out footer */
    fwrite(pFlatBuffer->pData, 1, dcResults.produced, dstFile);
```

```
}
```
}

Because this session was created with `CPA_DC_DIR_COMBINED` it can also be used to decompress data.
因为此会话是用 CPA _ DC _目录_组合创建的，所以它也可用于解压缩数据。

The Stateful Decompression Operation demonstrates looping through a file, reading the compressed data, invoking the data decompress operation, and writing the results to the output file. In this case, the overflow condition has to be considered.
有状态解压缩操作演示了遍历文件、读取压缩数据、调用数据解压缩操作，以及将结果写入输出文件。在这种情况下，必须考虑溢出条件。

Listing 67. Perform Stateful Decompression Operation
清单 67。执行有状态解压缩操作

```
pBufferListSrc->pBuffers->dataLenInBytes = 0;
while ((!feof(srcFile)) && (CPA_STATUS_SUCCESS == status))
{
    /* read from file into src buffer */
    pBufferListSrc->pBuffers->pData = pSrcBuffer;
    pBufferListSrc->pBuffers->dataLenInBytes +=
    fread(
        pSrcBuffer + pBufferListSrc->pBuffers-
        >dataLenInBytes, 1,
        SAMPLE_BUFF_SIZE - pBufferListSrc->pBuffers->dataLenInBytes,
        srcFile);
    if (pBufferListSrc->pBuffers->dataLenInBytes < SAMPLE_BUFF_SIZE)
    {
        /* FLUSH FINAL flag must be set for last request */
        opData.flushFlag = CPA_DC_FLUSH_FINAL;
    }
    else
    {
        /* FLUSH SYNC flag must be set for intermediate requests */
        opData.flushFlag = CPA_DC_FLUSH_SYNC;
    }
    do
    {
        status =
            cpaDcDecompressData2( dc
            InstHandle,
            sessionHdl,
            pBufferListSrc, /* source buffer list */
            pBufferListDst, /* destination buffer list */
            &opData,
            &dcResults, /* results structure */
            NULL);
        if (CPA_STATUS_SUCCESS != status)
        {
```

```
pBufferListSrc->pBuffers->dataLenInBytes = 0;
while ((!feof(srcFile)) && (CPA_STATUS_SUCCESS == status))
{
    /* read from file into src buffer */ pBufferListSrc-
    >pBuffers->pData = pSrcBuffer; pBufferListSrc->pBuffers-
    >dataLenInBytes += fread(
        pSrcBuffer + pBufferListSrc->pBuffers->dataLenInBytes, 1,
        SAMPLE_BUFF_SIZE - pBufferListSrc->pBuffers->dataLenInBytes, srcFile);
    if (pBufferListSrc->pBuffers->dataLenInBytes < SAMPLE_BUFF_SIZE)
    {
        /* FLUSH FINAL flag must be set for last request */ opData.flushFlag =
        CPA_DC_FLUSH_FINAL;
    }
    else
    {
        /* FLUSH SYNC flag must be set for intermediate requests */ opData.flushFlag
        = CPA_DC_FLUSH_SYNC;
    }
    do
    {
        status = cpaDcDecompressData2(
            dcInstHandle,
            sessionHdl,
            pBufferListSrc, /* source buffer list */ pBufferListDst, /*
            destination buffer list */ &opData,
            &dcResults, /* results structure */ NULL);
        if (CPA_STATUS_SUCCESS != status)
        {
            PRINT_ERR(
                "cpaDcDecompressData2 failed. (status = %d)\n", status);
    break;
}

/*
 * We now check the results - in decompress direction the
 * output buffer may overflow
 */
if ((dcResults.status != CPA_DC_OK) &&
    (dcResults.status != CPA_DC_OVERFLOW))
{
    PRINT_ERR(
        "Results status not as expected (status = %d)\n",
        dcResults.status);
        status =
    CPA_STATUS_FAIL; break;
}

/* The gzip file generated by Deflate algorithm has an
 * 8-byte
 * footer, containing a CRC-32 checksum and the length of
 * the
 * original uncompressed data. The 'endOfLastBlock' flag
 * tells
 * if we have processed the last data block. Break the loop
 * here, otherwise it will keep on reading gzip file.
 */
if (CPA_TRUE == dcResults.endOfLastBlock)
{
    break;
}

if (dcResults.consumed <=
    pBufferListSrc->pBuffers->dataLenInBytes)
{
    pBufferListSrc->pBuffers->dataLenInBytes -=
        dcResults.consumed;
    pBufferListSrc->pBuffers->pData += dcResults.consumed;
}
else
{
    pBufferListSrc->pBuffers->dataLenInBytes = 0;
}

if (dcResults.consumed == 0 &&
    pBufferListSrc->pBuffers->dataLenInBytes > 0)
```

```
        break;
    }

    /*
     * We now check the results - in decompress direction the
     * output buffer may overflow
     */
    if ((dcResults.status != CPA_DC_OK) &&
        (dcResults.status != CPA_DC_OVERFLOW))
    {
        PRINT_ERR(
            "Results status not as expected (status = %d)\n",
            dcResults.status);
            status = CPA_STATUS_FAIL;
        break;
    }

    /* The gzip file generated by Deflate algorithm has an
     * 8-byte
     * footer, containing a CRC-32 checksum and the length of
     * the
     * original uncompressed data. The 'endOfLastBlock' flag
     * tells
     * if we have processed the last data block. Break the loop
     * here, otherwise it will keep on reading gzip file.
     */
    if (CPA_TRUE == dcResults.endOfLastBlock)
    {
        break;
    }

    if (dcResults.consumed <=
        pBufferListSrc->pBuffers->dataLenInBytes)
    {
        pBufferListSrc->pBuffers->dataLenInBytes -= dcResults.consumed;
        pBufferListSrc->pBuffers->pData += dcResults.consumed;
    }
    else
    {
        pBufferListSrc->pBuffers->dataLenInBytes = 0;
    }

    if (dcResults.consumed == 0 &&
        pBufferListSrc->pBuffers->dataLenInBytes > 0)
```

```
            memcpy(pSrcBuffer,
                pBufferListSrc->pBuffers->pData,
                pBufferListSrc->pBuffers-
                >dataLenInBytes);
            break;
        }
    } while (pBufferListSrc->pBuffers->dataLenInBytes != 0 ||
```

```
            memcpy(pSrcBuffer,
                pBufferListSrc->pBuffers->pData, pBufferListSrc-
                >pBuffers->dataLenInBytes);
            break;
        }
    } while (pBufferListSrc->pBuffers->dataLenInBytes != 0 ||
             dcResults.status == CPA_DC_OVERFLOW);
```

Once all operations on this session have been completed, the session is torn down using the Remove Stateful Session in Listing 68.

完成此会话上的所有操作后，使用中的“删除有状态会话”拆除会话 Listing 68

Listing 68. Remove Stateful Session

清单 68。删除有状态会话

```
sessionStatus = cpaDcRemoveSession(dcInstHandle, sessionHdl);
```

```
session status = cpaDcRemoveSession(dcInstHandle, session HDL);
```

Query statistics at this point, which can be useful for debugging.

Finally, clean up by freeing up memory, stopping the instance, etc.

此时查询统计信息，这对调试很有用。最后，通过释放内存、停止实例等方式进行

清理。

4.2.3 Sample – Stateless Data Compression

4.2.4 示例 – 无状态数据压缩

This example demonstrates the usage of the asynchronous API, specifically using this API to perform a compression operation. It compresses a data buffer through a stateless session using the deflate compress algorithm with dynamic Huffman trees.

这个例子演示了异步 API 的用法，特别是使用这个 API 来执行压缩操作。它使用 deflate 压缩算法和动态霍夫曼树，通过无状态会话压缩数据缓冲区。

The example below compresses a block of data into a compressed block.

These samples are located in `/dc/stateless_sample`

以下示例将数据块压缩成压缩块。这些示例位于/dc/stateless_sample 中

In this example, dynamic Huffman trees are used. The instance can be queried to ensure dynamic Huffman trees are supported, and if an instance-specific buffer is required to perform a dynamic Huffman tree deflate request.
在这个例子中，使用了动态霍夫曼树。可以查询实例以确保支持动态霍夫曼树，以及是否需要特定于实例的缓冲区来执行动态霍夫曼树紧缩请求。

Listing 69. Querying and Starting a Compression Instance
清单 69。查询和启动压缩实例

```
status = cpaDcQueryCapabilities(dcInstHandle,
&cap); if (status != CPA_STATUS_SUCCESS) {
    return status;
}

if (!cap.statelessDeflateCompression ||
!cap.statelessDeflateDecompression ||
    !cap.checksumAdler32 ||
    !cap.dynamicHuffman) {
    PRINT_DBG("Error: Unsupported functionality\
    n"); return CPA_STATUS_FAIL;
}

if (cap.dynamicHuffmanBufferReq) {
```

```
status = cpaDcQueryCapabilities(dcInstHandle, &cap); if
(status != CPA_STATUS_SUCCESS) {
    return status;
}

if (!cap.statelessDeflateCompression ||
!cap.statelessDeflateDecompression ||
    !cap.checksumAdler32 ||
    !cap.dynamicHuffman) {
    PRINT_DBG("Error: Unsupported functionality\n"); return
    CPA_STATUS_FAIL;
}

if (cap.dynamicHuffmanBufferReq) {
    if (CPA_STATUS_SUCCESS == status) {
        status =
            cpaDcGetNumIntermediateBuffers(dcInstHandle,
&numInterBuffLists);
    }
    if (CPA_STATUS_SUCCESS == status && 0 != numInterBuffLists)
        { status = PHYS_CONTIG_ALLOC(&bufferInterArray,
                                   numInterBuffLists *
sizeof(CpaBufferList *));
    }
    for (bufferNum = 0; bufferNum < numInterBuffLists; bufferNum++)
        { if (CPA_STATUS_SUCCESS == status) {
            status = PHYS_CONTIG_ALLOC(&bufferInterArray[bufferNum],
                                       sizeof(CpaBufferList));
        }
        if (CPA_STATUS_SUCCESS == status)
          { status = PHYS_CONTIG_ALLOC(
                &bufferInterArray[bufferNum]->pPrivateMetaData,
buffMetaSize);
        }
       if (CPA_STATUS_SUCCESS == status) {
            status = PHYS_CONTIG_ALLOC(&bufferInterArray[bufferNum]-
>pBuffers,
                                       sizeof(CpaFlatBuffer));
        }
       if (CPA_STATUS_SUCCESS == status) {
            /* Implementation requires an intermediate buffer
approximately
                       twice the size of the output buffer */
            status =
                PHYS_CONTIG_ALLOC(&bufferInterArray[bufferNum]-
                >pBuffers-
>pData,
                                  2 * SAMPLE_MAX_BUFF);
            bufferInterArray[bufferNum]->numBuffers = 1;
            bufferInterArray[bufferNum]->pBuffers->dataLenInBytes
            =
                2 * SAMPLE_MAX_BUFF;
        }

    } /* End numInterBuffLists */
}

if (CPA_STATUS_SUCCESS == status) {
    /*
     * Set the address translation function for the instance
```

```
        if (CPA_STATUS_SUCCESS == status) { status =
                cpaDcGetNumIntermediateBuffers(dcInstHandle, &numInterBuffLists);
        }
        if (CPA_STATUS_SUCCESS == status && 0 != numInterBuffLists) { status =
            PHYS_CONTIG_ALLOC(&bufferInterArray,
                                                numInterBuffLists *
sizeof(CpaBufferList *));
        }
        for (bufferNum = 0; bufferNum < numInterBuffLists; bufferNum++) { if
             (CPA_STATUS_SUCCESS == status) {
                status = PHYS_CONTIG_ALLOC(&bufferInterArray[bufferNum],
                                                     sizeof(CpaBufferList));
            }
              if (CPA_STATUS_SUCCESS == status) {
                  status = PHYS_CONTIG_ALLOC(
                     &bufferInterArray[bufferNum]->pPrivateMetaData,
buffMetaSize);
            }
              if (CPA_STATUS_SUCCESS == status) {
                status = PHYS_CONTIG_ALLOC(&bufferInterArray[bufferNum]-
>pBuffers,
                                                     sizeof(CpaFlatBuffer));
            }
              if (CPA_STATUS_SUCCESS == status) {
                /* Implementation requires an intermediate buffer approximately
                             twice the size of the output buffer */
                status =
                     PHYS_CONTIG_ALLOC(&bufferInterArray[bufferNum]->pBuffers-
>pData,
                                              2 * SAMPLE_MAX_BUFF);
                bufferInterArray[bufferNum]->numBuffers = 1;
                bufferInterArray[bufferNum]->pBuffers->dataLenInBytes =
                     2 * SAMPLE_MAX_BUFF;
            }

        } /* End numInterBuffLists */
}

if (CPA_STATUS_SUCCESS == status) {
        /*
         * Set the address translation function for the instance
         */
        status = cpaDcSetAddressTranslation(dcInstHandle, sampleVirtToPhys);
```

```
}

if (CPA_STATUS_SUCCESS == status) {
    /* Start DataCompression component */
    PRINT_DBG("cpaDcStartInstance\n");
    status =
        cpaDcStartInstance(dcInstHandle, numInterBuffLists,
bufferInterArray);
```

```
}

if (CPA_STATUS_SUCCESS == status) {
    /* Start DataCompression component */
    PRINT_DBG("cpaDcStartInstance\n"); status =
        cpaDcStartInstance(dcInstHandle, numInterBuffLists, bufferInterArray);
}
```

The create and initialize stateless session demonstrates the sequence of calls to be made to the API to create a session. To establish a session: describing the session, determining how much session memory is required, and then invoke the session initialization function `cpaDcInitSession`.

创建和初始化无状态会话演示了调用 API 创建会话的顺序。建立会话:描述会话，确定需要多少会话内存，然后调用会话初始化函数 cpaDcInitSession。

Listing 70. Create and Initialize Stateless Session

清单 70。创建和初始化无状态会话

```
sd.compLevel = CPA_DC_L4;
sd.compType = CPA_DC_DEFLATE;
sd.huffType = CPA_DC_HT_FULL_DYNAMIC;
/* If the implementation supports it, the session will be configured
 * to select static Huffman encoding over dynamic Huffman as
 * the static encoding will provide better compressibility.
 */
if (cap.autoSelectBestHuffmanTree) {
    sd.autoSelectBestHuffmanTree = CPA_TRUE;
} else {
    sd.autoSelectBestHuffmanTree = CPA_FALSE;
}
sd.sessDirection =
CPA_DC_DIR_COMBINED; sd.sessState =
CPA_DC_STATELESS;
#if (CPA_DC_API_VERSION_NUM_MAJOR == 1 && CPA_DC_API_VERSION_NUM_MINOR
< 6)
sd.deflateWindowSize =
7; #endif
sd.checksum = CPA_DC_ADLER32;
/* Determine size of session context to allocate */
PRINT_DBG("cpaDcGetSessionSize\n");
status = cpaDcGetSessionSize(dcInstHandle, &sd, &sess_size, &ctx_size);
```

```
sd.compLevel = CPA_DC_L4; sd.compType =
CPA_DC_DEFLATE;
sd.huffType = CPA_DC_HT_FULL_DYNAMIC;
/* If the implementation supports it, the session will be configured
 * to select static Huffman encoding over dynamic Huffman as
 * the static encoding will provide better compressibility.
 */
if (cap.autoSelectBestHuffmanTree) {
    sd.autoSelectBestHuffmanTree = CPA_TRUE;
} else {
    sd.autoSelectBestHuffmanTree = CPA_FALSE;
}
sd.sessDirection = CPA_DC_DIR_COMBINED;
sd.sessState = CPA_DC_STATELESS;
#if (CPA_DC_API_VERSION_NUM_MAJOR == 1 && CPA_DC_API_VERSION_NUM_MINOR < 6)
sd.deflateWindowSize = 7;
#endif
sd.checksum = CPA_DC_ADLER32;
/* Determine size of session context to allocate */ PRINT_DBG("cpaDcGetSessionSize\n");
status = cpaDcGetSessionSize(dcInstHandle, &sd, &sess_size, &ctx_size);
}
```

```
if (CPA_STATUS_SUCCESS == status) {
    PRINT_DBG("cpaDcInitSession\n");
    status = cpaDcInitSession(
        dcInstHandle,
        sessionHdl,  /* session memory */
        &sd,         /* session setup data
        */
        NULL,        /* pContexBuffer not required for
stateless operations */
```

```
if (CPA_STATUS_SUCCESS == status) {
    PRINT_DBG("cpaDcInitSession\n"); status
    = cpaDcInitSession(
        dcInstHandle,
        sessionHdl,  /* session memory */ &sd,
                     /* session setup data */
        NULL,        /* pContexBuffer not required for stateless
operations */
        dcCallback); /* callback function */
```

Source and destination buffers are allocated in a similar way to the stateful example above.

源缓冲区和目的缓冲区的分配方式与上面的有状态示例类似。

Perform Operation: This listing demonstrates invoking the data compress operation, in the stateless case.
执行操作:这个清单演示了在无状态的情况下调用数据压缩操作。

Listing 71. Data Plane Remove Compression Session
清单 71。数据平面删除压缩会话

```
sessionStatus = cpaDcRemoveSession(dcInstHandle, sessionHdl);
```

```
session status = cpaDcRemoveSession(dcInstHandle, session HDL);
```

8.5 Sample – Stateless Data Compression Using Multiple Compress Operations

8.6 示例 – 使用多个压缩操作的无状态数据压缩

This example demonstrates the use of the asynchronous API: specifically, using this API to perform a compression operation. It compresses a data buffer using multiple stateless compression API requests and maintains length and checksum information across the multiple requests without the overhead of maintaining full history information as used in a stateful operation.

这个例子演示了异步 API 的用法:具体来说，就是使用这个 API 来执行压缩操作。它使用多个无状

态压缩 API 请求来压缩数据缓冲区，并跨多个请求维护长度和校验和信息，而没有维护有状态操作中使用的完整历史信息的开销。

The samples are located in: /dc/stateless_multi_op_checksum_sample
样本位于:/DC/stateless _ multi _ op _ checksum _ sample

In this sample, session creation is the same as for regular stateless operation. Refer to the previous sample described in Section 4.3, Sample – Stateless Data Compression Using Multiple Compress Operations for details.

在这个示例中，会话创建与常规的无状态操作相同。请参考第节中描述的前一个示例 4.3Sample – Stateless Data CompressionUsing Multiple Compress Operations

Perform Operation: This listing demonstrates the invoking of the data compress operation in the stateless case while maintaining checksum information across multiple compress operations. The key points to note are:
执行操作:这个清单演示了在无状态情况下调用数据压缩操作，同时跨多个压缩操作维护校验和信息。需要注意的要点是:

The initial value of dcResults.checksum is set to 0 for CRC32 or set to 1 for Adler32 when invoking the first compress or decompress operation for a data set.
当调用数据集的第一次压缩或解压缩操作时，CRC32 的 dcResults.checksum 初始值设置为 0，Adler32 的初始值设置为 1。

Listing 72. Setting the Initial Value of the Checksum
清单 72。设置校验和的初始值

```
if (sd.checksum == CPA_DC_ADLER32) {
if (sd.checksum == CPA_DC_ADLER32) {
    /* Initialize checksum to 1 for Adler32 */
    /* Initialize checksum to 1 for Adler32 */
    dcResults.checksum = 1;
    dcResults.checksum = 1;
} else {
} else {
    /* Initialize checksum to 0 for CRC32 */
    /* Initialize checksum to 0 for CRC32 */
}
}
```

The value of dcResults.checksum when invoking a subsequent compress operation for a data set is set to the dcResults. Checksum value returned from the previous compress operation on that data set.
调用数据集的后续压缩操作时，dcResults.checksum 的值被设置为 dcResults。从该数据集上一次压缩操作返回的校验和值。

8.7 Sample – Data Compression Data Plane API
8.8 样本 – 数据压缩数据平面 API

This example demonstrates the usage of the data plane data compression API to perform a compression operation. It compresses a data buffer via a stateless session using the deflate compress algorithm with dynamic Huffman trees. This example is simplified to demonstrate the basics of how to use the API and how to build the structures required. This example does not demonstrate the optimal way to use the API to get maximum performance for a particular implementation. Refer to Table 1 Implementation Specific Documentation and performance sample code for a guide on how to use the API for best performance.

此示例演示了如何使用数据平面数据压缩 API 来执行压缩操作。它使用 deflate 压缩算法和动态霍夫曼树，通过无状态会话压缩数据缓冲区。这个例子被简化以演示如何使用 API 以及如何构建所需结构的基础知识。这个例子没有展示使用 API 来获得特定实现的最佳性能的最佳方式。涉及 Table 1

These samples are located in /dc/dc_dp_sample
这些示例位于/dc/dc_dp_sample 中

The data plane data compression API is used in a similar way to the data plane-symmetric cryptographic API:

数据平面数据压缩 API 的使用方式类似于数据平面对称加密 API:

Data compression service instances are queried and started in the same way and using the same functions as before (see Listing 1 and Listing 68).
数据压缩服务实例的查询和启动方式与以前相同，使用的功能也与以前相同(请参见 Listing 1 Listing 68

Listing 73. Register Compression Callback Function
清单 73。寄存器压缩回调函数

This listing registers a callback function for the data compression instance.
这个清单为数据压缩实例注册了一个回调函数。

```
status = cpaDcDpRegCbFunc(dcInstHandle, dcDpCallback);
```

```
status = cpaDcDpRegCbFunc(dcInstHandle, dcDpCallback);
```

Next, create and initialize a session.

接下来，创建并初始化一个会话。

Listing 74. Create and Initialize Compression Data Plane Session
清单 74。创建并初始化压缩数据平面会话

```
if (CPA_STATUS_SUCCESS == status) {
    sd.compLevel = CPA_DC_L4;
    sd.compType = CPA_DC_DEFLATE;
    sd.huffType = CPA_DC_HT_FULL_DYNAMIC;
    /* If the implementation supports it, the session will be configured
     * to select static Huffman encoding over dynamic Huffman as
     * the static encoding will provide better compressibility.
     */
    if (cap.autoSelectBestHuffmanTree) {
        sd.autoSelectBestHuffmanTree = CPA_TRUE;
    } else {
        sd.autoSelectBestHuffmanTree = CPA_FALSE;
    }
    sd.sessDirection = CPA_DC_DIR_COMBINED;
    sd.sessState = CPA_DC_STATELESS;
#if (CPA_DC_API_VERSION_NUM_MAJOR == 1 && CPA_DC_API_VERSION_NUM_MINOR < 6)
```

```
    sd.deflateWindowSize =
7; #endif
   sd.checksum = CPA_DC_CRC32;

    /* Determine size of session context to allocate */
    PRINT_DBG("cpaDcGetSessionSize\n");
    status = cpaDcGetSessionSize(dcInstHandle, &sd, &sess_size,
&ctx_size);
}

if (CPA_STATUS_SUCCESS == status)
                {
    /* Allocate session memory */
    status = PHYS_CONTIG_ALLOC(&sessionHdl, sess_size);
}

/* Initialize the Stateless session
*/ if (CPA_STATUS_SUCCESS == status)
{
    PRINT_DBG("cpaDcDpInitSession\n");
    status = cpaDcDpInitSession(dcInstHandle,
                                sessionHdl, /* session memory */
```

```
    sd.deflateWindowSize = 7;
#endif
      sd.checksum = CPA_DC_CRC32;

    /* Determine size of session context to allocate */
    PRINT_DBG("cpaDcGetSessionSize\n");
    status = cpaDcGetSessionSize(dcInstHandle, &sd, &sess_size, &ctx_size);
}

  if (CPA_STATUS_SUCCESS == status) {
    /* Allocate session memory */
    status = PHYS_CONTIG_ALLOC(&sessionHdl, sess_size);
}

/* Initialize the Stateless session */ if
(CPA_STATUS_SUCCESS == status) {
    PRINT_DBG("cpaDcDpInitSession\n");
    status = cpaDcDpInitSession(dcInstHandle,
                                sessionHdl, /* session memory */ &sd);
                                            /* session setup data */
}
```

Listing 75. Setup Source Buffer

清单 75。设置源缓冲区

In this example, input and output data is stored in a scatter gather list. The source and destination buffers are described using the `CpaPhysBufferList` structure. In this example the allocation (which needs to be 8-byte aligned) and setup of the source buffer is shown. The destination buffers can be allocated and set up in a similar way.

在此示例中，输入和输出数据存储在分散收集列表中。源缓冲区和目的缓冲区是用 CpaPhysBufferList 结构描述的。在本例中，显示了源缓冲区的分配(需要 8 字节对齐)和设置。可以用类似的方式分配和设置目的缓冲器。

```
numBuffers = 2;
/* Size of CpaPhysBufferList and array of CpaPhysFlatBuffers */
bufferListMemSize =
    sizeof(CpaPhysBufferList) + (numBuffers *
    sizeof(CpaPhysFlatBuffer));

/* Allocte 8-byte aligned source buffer List */
status = PHYS_CONTIG_ALLOC_ALIGNED(&pBufferListSrc,
bufferListMemSize, 8);
if (CPA_STATUS_SUCCESS == status) {
    /* Allocate first data buffer to hold half the data */
    status = PHYS_CONTIG_ALLOC(&pSrcBuffer, (sizeof(sampleData)) / 2);
}
if (CPA_STATUS_SUCCESS == status) {
    /* Allocate second data buffer to hold half the data */
    status = PHYS_CONTIG_ALLOC(&pSrcBuffer2, (sizeof(sampleData)) / 2);
}
if (CPA_STATUS_SUCCESS == status) {
```

```
numBuffers = 2;
/* Size of CpaPhysBufferList and array of CpaPhysFlatBuffers */ bufferListMemSize =
    sizeof(CpaPhysBufferList) + (numBuffers * sizeof(CpaPhysFlatBuffer));
/* Allocte 8-byte aligned source buffer List */
status = PHYS_CONTIG_ALLOC_ALIGNED(&pBufferListSrc, bufferListMemSize, 8);
if (CPA_STATUS_SUCCESS == status) {
    /* Allocate first data buffer to hold half the data */
    status = PHYS_CONTIG_ALLOC(&pSrcBuffer, (sizeof(sampleData)) / 2);
}
if (CPA_STATUS_SUCCESS == status) {
    /* Allocate second data buffer to hold half the data */
    status = PHYS_CONTIG_ALLOC(&pSrcBuffer2, (sizeof(sampleData)) / 2);
}
if (CPA_STATUS_SUCCESS == status) {
    /* copy source into buffer */
    memcpy(pSrcBuffer, sampleData, sizeof(sampleData) / 2);
    memcpy(pSrcBuffer2,
           &(sampleData[sizeof(sampleData) / 2]),
           sizeof(sampleData) / 2);

    /* Build source bufferList */
    pBufferListSrc->numBuffers =
    2;
    pBufferListSrc->flatBuffers[0].dataLenInBytes = sizeof(sampleData)
    /
2;
    pBufferListSrc->flatBuffers[0].bufferPhysAddr =
        sampleVirtToPhys(pSrcBuffer);
    pBufferListSrc->flatBuffers[1].dataLenInBytes = sizeof(sampleData)
    /
2;
```

```
    memcpy(pSrcBuffer, sampleData, sizeof(sampleData) / 2); memcpy(pSrcBuffer2,
           &(sampleData[sizeof(sampleData) / 2]),
           sizeof(sampleData) / 2);

    /* Build source bufferList */
    pBufferListSrc->numBuffers = 2;
    pBufferListSrc->flatBuffers[0].dataLenInBytes = sizeof(sampleData) /
2;
    pBufferListSrc->flatBuffers[0].bufferPhysAddr = sampleVirtToPhys(pSrcBuffer);
    pBufferListSrc->flatBuffers[1].dataLenInBytes = sizeof(sampleData) /
2;
    pBufferListSrc->flatBuffers[1].bufferPhysAddr = sampleVirtToPhys(pSrcBuffer2);
```

The operational data in this case is:

这种情况下的操作数据是:

Listing 76. Compression Data Plane Operational Data
清单 76。压缩数据平面操作数据

```
/* Allocate memory for operational data. Note this needs to be
 * 8-byte aligned, contiguous, resident in DMA-accessible
 * memory.
 */
status = PHYS_CONTIG_ALLOC_ALIGNED(&pOpData, sizeof(CpaDcDpOpData), 8);
}
if (CPA_STATUS_SUCCESS == status) {
    pOpData->bufferLenToCompress =
    sizeof(sampleData); pOpData->bufferLenForData =
    sizeof(sampleData); pOpData->dcInstance =
    dcInstHandle;
    pOpData->pSessionHandle = sessionHdl;
    pOpData->srcBuffer =
```

```
/* Allocate memory for operational data. Note this needs to be
 * 8-byte aligned, contiguous, resident in DMA-accessible
 * memory.
 */
status = PHYS_CONTIG_ALLOC_ALIGNED(&pOpData, sizeof(CpaDcDpOpData), 8);
}
if (CPA_STATUS_SUCCESS == status) {
    pOpData->bufferLenToCompress = sizeof(sampleData);
    pOpData->bufferLenForData = sizeof(sampleData); pOpData-
>dcInstance = dcInstHandle;
    pOpData->pSessionHandle = sessionHdl;
    pOpData->srcBuffer = sampleVirtToPhys(pBufferListSrc);
```

This request is then enqueued and submitted on the instance.

然后，该请求在实例上排队并提交。

Listing 77. Data Plane Enqueue and Submit
清单 77。数据平面排队并提交

```
status = cpaDcDpEnqueueOp(pOpData, CPA_TRUE);
status = cpaDcDpEnqueueOp(pOpData, CPA _ TRUE);
```

- After possibly doing other work (e.g., enqueuing and submitting more requests), the Application can poll for responses that invoke the callback function registered
- 在可能完成其他工作(例如，排队和提交更多请求)之后，应用程序可以轮询调用已注册的回调函数的响应

with the instance. Refer to Table 1 Implementation Specific Documentation on the implementations polling functions.
使用实例。涉及 Table 1

- Once all requests associated with a session have been completed, the session can be removed.
- 一旦完成了与会话相关联的所有请求，就可以删除该会话。

Listing 78. Data Plane Remove Compression Session
清单 78。数据平面删除压缩会话

Clean up by freeing up memory, stopping the instance, etc. using this command:
通过释放内存、停止实例等方式进行清理。使用此命令：

```
sessionStatus = cpaDcDpRemoveSession(dcInstHandle, sessionHdl);
```

```
session status = cpaDcDpRemoveSession(dcInstHandle, session HDL);
```

8.9 Sample - Chained Hash and Stateless Compression

8.10 样本链散列和无状态压缩

This example demonstrates the use of the asynchronous API, specifically, using the data compression chain API to perform chained hash and stateless compression operations. It performs a sha256 hash on the sample text and then compresses the sample text through a stateless session using the deflate compress algorithm with static Huffman trees.

这个例子演示了异步 API 的使用，特别是使用数据压缩链 API 来执行链式散列和无状态压缩操作。它对样本文本执行 sha256 散列，然后使用 deflate 压缩算法和静态霍夫曼树通过无状态会话压缩样本文本。

These samples are located in `/dc/chaining_sample`
这些示例位于/dc/chaining_sample 中

Listing 79. Querying and Starting a Compression Instance
清单 79。查询和启动压缩实例

```
/*
 * In this simplified version of instance discovery, we discover
 * exactly one instance of a data compression service.
 */
sampleDcGetInstance(&dcInstHandle
); if (dcInstHandle == NULL)
{
    PRINT_ERR("Get instance failed\
    n"); return CPA_STATUS_FAIL;
}

/* Query Capabilities */
PRINT_DBG("cpaDcQueryCapabilities\n");
status = cpaDcQueryCapabilities(dcInstHandle,
&cap); if (status != CPA_STATUS_SUCCESS)
```

```
/*
 * In this simplified version of instance discovery, we discover
 * exactly one instance of a data compression service.
 */ sampleDcGetInstance(&dcInstHandle);
if (dcInstHandle == NULL)
{
    PRINT_ERR("Get instance failed\n");
    return CPA_STATUS_FAIL;
}

/* Query Capabilities */ PRINT_DBG("cpaDcQueryCapabilities\
n");
status = cpaDcQueryCapabilities(dcInstHandle, &cap); if
(status != CPA_STATUS_SUCCESS)
{
        "instance.\n");
    PRINT_ERR("Please ensure StorageEnabled=1 in the device
    configuration
"
              "file.\n");
    return
    CPA_STATUS_FAIL;
}

if (!cap.statelessDeflateCompression || !cap.checksumCRC32 ||
    !cap.checksumAdler32)
{
    PRINT_ERR("Error: Unsupported functionality\
    n"); return CPA_STATUS_FAIL;
}

if (CPA_STATUS_SUCCESS == status)
{
    /* Set the address translation function for the instance */
    status = cpaDcSetAddressTranslation(dcInstHandle,
    sampleVirtToPhys);
}

if (CPA_STATUS_SUCCESS == status)
{
    /* Start static data compression component */
    PRINT_DBG("cpaDcStartInstance\n");
```

```
        "instance.\n");
    PRINT_ERR("Please ensure StorageEnabled=1 in the device configuration
"
                "file.\n");
    return CPA_STATUS_FAIL;
}

if (!cap.statelessDeflateCompression || !cap.checksumCRC32 ||
    !cap.checksumAdler32)
{
    PRINT_ERR("Error: Unsupported functionality\n"); return
    CPA_STATUS_FAIL;
}

if (CPA_STATUS_SUCCESS == status)
{
    /* Set the address translation function for the instance */
    status = cpaDcSetAddressTranslation(dcInstHandle, sampleVirtToPhys);
}

if (CPA_STATUS_SUCCESS == status)
{
    /* Start static data compression component */ PRINT_DBG("cpaDcStartInstance\n");
    status = cpaDcStartInstance(dcInstHandle, 0, NULL);
}
```

Listing 80. Create and Initialize Session Hash and Compression

清单 80。创建并初始化会话哈希和压缩

```
if (CPA_STATUS_SUCCESS == status)
{
    /*
     * If the instance is polled start the polling thread. Note that
     * how the polling is done is implementation-dependant.
     */
    sampleDcStartPolling(dcInstHandle);
    /*
     * We now populate the fields of the session operational data
and create
     * the session.  Note that the size required to store a session is
     * implementation-dependent, so we query the API first to determine
how
     * much memory to allocate, and then allocate that memory.
     */
```

```
if (CPA_STATUS_SUCCESS == status)
{
    /*
     * If the instance is polled start the polling thread. Note that
     * how the polling is done is implementation-dependant.
     */ sampleDcStartPolling(dcInstHandle);
    /*
     * We now populate the fields of the session operational data and create
     * the session.  Note that the size required to store a session is
     * implementation-dependent, so we query the API first to determine
how
     * much memory to allocate, and then allocate that memory.
     */

    //<snippet name="initSession">
    /* Initialize compression session data */
    dcSessionData.compLevel = CPA_DC_L1;
    dcSessionData.compType = CPA_DC_DEFLATE;
    dcSessionData.huffType = CPA_DC_HT_STATIC;
    dcSessionData.autoSelectBestHuffmanTree = CPA_FALSE;
    dcSessionData.sessDirection = CPA_DC_DIR_COMPRESS;
    dcSessionData.sessState = CPA_DC_STATELESS;
    dcSessionData.checksum = CPA_DC_CRC32;
    /*初始化压缩会话数据*/dcsessiondata . com level = CPA _ DC _ L1;
    dcsessiondata . comptype = CPA _ DC _ DEFLATE; dcsessiondata . huff
    type = CPA _ DC _ HT _ STATIC; dcsessiondata .
    autoselectbesthufmantree = CPA _ FALSE; dcsessiondata . sess
    direction = CPA _ DC _目录_压缩; dcsessiondata . sesstate = CPA _
    DC _无状态; dcsessiondata . checksum = CPA _ DC _ CRC32;


    /* Initialize crypto session data */
    cySessionData.sessionPriority = CPA_CY_PRIORITY_NORMAL;
    /*初始化加密会话数据*/cysession data . session PRIORITY = CPA _ CY _
    PRIORITY _ NORMAL;
    /* Hash operation on the source data */
    cySessionData.symOperation = CPA_CY_SYM_OP_HASH;
    /*对源数据的哈希运算*/cysession data . sym operation = CPA _ CY _ SYM _ OP
    _ Hash;
    cySessionData.hashSetupData.hashAlgorithm = CPA_CY_SYM_HASH_SHA256;
    cySessionData.hashSetupData.hashMode = CPA_CY_SYM_HASH_MODE_PLAIN;
    cySessionData.hashSetupData.digestResultLenInBytes =
    cysession data . hashsetupdata . hashalgorithm = CPA _ CY _ SYM _哈希_
    SHA256cysession data . hashsetupdata . HASH MODE = CPA _ CY _ SYM _哈希_模式_平原;
    cysession data . hashsetupdata . digestresultleninbytes =

GET_HASH_DIGEST_LENGTH(cySessionData.hashSetupData.hashAlgorithm);
GET _ HASH _ DIGEST _ LENGTH(cysession data . hashsetupdata . hashalgorithm);
    /* Place the digest result in a buffer unrelated to srcBuffer */
    cySessionData.digestIsAppended = CPA_FALSE;
    /*将摘要结果放在与src buffer */cysession data . digestisappended = CPA _ FALSE无关的
    缓冲区中;
    /* Generate the digest */
    cySessionData.verifyDigest = CPA_FALSE;
    /*生成摘要*/cysession data . verify digest = CPA
    _ FALSE;


    /* Initialize chaining session data - hash + compression
    /*初始化链接会话数据-哈希+压缩
     * chain operation */
```

```
        *连锁经营*/
    chainSessionData[0].sessType = CPA_DC_CHAIN_SYMMETRIC_CRYPTO;
    chainSessionData[0].pCySetupData = &cySessionData;
    chainSessionData[1].sessType = CPA_DC_CHAIN_COMPRESS_DECOMPRESS;
    chainSessionData[1].pDcSetupData = &dcSessionData;

    chainSessionData[0]。sess type = CPA _ DC _连锁_对称_加密；
    chainSessionData[0]。pCySetupData = & cySessionDatachainSessionData[1]。sess type =
    CPA _ DC _连锁_压缩_解压缩；chainSessionData[1]。pDcSetupData = & dcSessionData


    /* Determine size of session context to allocate */
    PRINT_DBG("cpaDcChainGetSessionSize\n");

    /*确定要分配的会话上下文的大小*/PRINT _ DBG(" cpadchaingetsessionsize \ n ")；
    status = cpaDcChainGetSessionSize(dcInstHandle,
    status = cpaDcChainGetSessionSize(dcInstHandle,
                                          CPA_DC_CHAIN_HASH_THEN_COMPRESS,
                                          NUM_SESSIONS_TWO,

                                          CPA _ DC _ CHAIN _ HASH _ THEN _
                                          COMPRESS，数量_会话_二，
                                          chainSessionData,
                                          &sess_size);
                                          chainSessionData，&
                                          sess _ size)；
}
}


if (CPA_STATUS_SUCCESS == status)
if (CPA_STATUS_SUCCESS == status)
{
{
    /* Allocate session memory */
    /*分配会话内存*/
    status = PHYS_CONTIG_ALLOC(&sessionHdl, sess_size);
    状态= PHYS _重叠_分配(&sessionHdl，sess _ size)；
}
}


/* Initialize the chaining session */
/*初始化链接会话*/
```

```
if (CPA_STATUS_SUCCESS == status)
{
    PRINT_DBG("cpaDcChainInitSession\n");
    status = cpaDcChainInitSession(dcInstHandle,
                                   sessionHdl,
                                   CPA_DC_CHAIN_HASH_THEN_COMPRESS,
                                   NUM_SESSIONS_TWO,
                                   chainSessionDat
                                   a, dcCallback);
```

```
if (CPA_STATUS_SUCCESS == status)
{
    PRINT_DBG("cpaDcChainInitSession\n");
    status = cpaDcChainInitSession(dcInstHandle,
                                   sessionHdl, CPA_DC_CHAIN_HASH_THEN_COMPRESS,
                                   NUM_SESSIONS_TWO,
                                   chainSessionData,
                                   dcCallback);
}
```

Note: `cySessionData.digestIsAppended` should be always set to `CPA_FALSE` as the digest must not appended in the end of output.

注意:cySessionData.digestIsAppended 应该始终设置为 CPA_FALSE，因为摘要不能附加在输出的末尾。

Listing 81. Chained Hash and Stateless Compression Memory Allocation
清单 81。链式散列和无状态压缩内存分配

```
status = cpaDcBufferListGetMetaSize(dcInstHandle, numBuffers,
&bufferMetaSize);
if (CPA_STATUS_SUCCESS != status)
{
    PRINT_ERR("Error get meta size\n");
    return CPA_STATUS_FAIL;
}
bufferSize = sampleDataSize;
if (CPA_STATUS_SUCCESS == status)
{
    status = dcChainBuildBufferList(&pBufferListSrc, numBuffers,
bufferSize, bufferMetaSize);
}

/* copy source data into buffer
*/ if (CPA_STATUS_SUCCESS ==
status)
{
    pFlatBuffer = (CpaFlatBuffer *)(pBufferListSrc +
    1); memcpy(pFlatBuffer->pData, sampleData,
    bufferSize);
}
/* Allocate destination buffer the four times as source buffer
```

```
status = cpaDcBufferListGetMetaSize(dcInstHandle, numBuffers, &bufferMetaSize);
if (CPA_STATUS_SUCCESS != status)
{
    PRINT_ERR("Error get meta size\n"); return
    CPA_STATUS_FAIL;
}
bufferSize = sampleDataSize;
if (CPA_STATUS_SUCCESS == status)
{
    status = dcChainBuildBufferList(&pBufferListSrc, numBuffers, bufferSize,
bufferMetaSize);
}

/* copy source data into buffer */ if
(CPA_STATUS_SUCCESS == status)
{
    pFlatBuffer = (CpaFlatBuffer *)(pBufferListSrc + 1);
    memcpy(pFlatBuffer->pData, sampleData, bufferSize);
}
/* Allocate destination buffer the four times as source buffer */ if
(CPA_STATUS_SUCCESS == status)
{
    status =
PHYS_CONTIG_ALLOC(&pDigestBuffer,
```

```
    status = PHYS_CONTIG_ALLOC(&pDigestBuffer,
GET_HASH_DIGEST_LENGTH(hashAlg));
```

Listing 82. Set Up Operational Data Hash and Compression

清单 82。设置操作数据哈希和压缩

```
dcOpData.flushFlag = CPA_DC_FLUSH_FINAL;
dcOpData.compressAndVerify = CPA_TRUE;
dcOpData.compressAndVerifyAndRecover =
CPA_TRUE;

cySymOpData.packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
cySymOpData.hashStartSrcOffsetInBytes = 0;
cySymOpData.messageLenToHashInBytes = bufferSize;
cySymOpData.pDigestResult = pDigestBuffer;

/* Set chaining operation data */
chainOpData[0].opType =
CPA_DC_CHAIN_SYMMETRIC_CRYPTO;
chainOpData[0].pCySymOp = &cySymOpData;
```

```
dcOpData.flushFlag = CPA_DC_FLUSH_FINAL;
dcOpData.compressAndVerify = CPA_TRUE;
dcOpData.compressAndVerifyAndRecover = CPA_TRUE;

cySymOpData.packetType = CPA_CY_SYM_PACKET_TYPE_FULL;
cySymOpData.hashStartSrcOffsetInBytes = 0; cySymOpData.messageLenToHashInBytes =
bufferSize; cySymOpData.pDigestResult = pDigestBuffer;
/* Set chaining operation data */ chainOpData[0].opType =
CPA_DC_CHAIN_SYMMETRIC_CRYPTO; chainOpData[0].pCySymOp =
&cySymOpData;
chainOpData[1].opType = CPA_DC_CHAIN_COMPRESS_DECOMPRESS; chainOpData[1].pDcOp =
&dcOpData;
```

Hash and stateless dynamic compression are also supported. Refer to Listing 64 and Listing 65 to add dynamic compression released buffers and session data.

还支持哈希和无状态动态压缩。涉及 Listing 64 Listing 65

Note: Hash algorithms are not limited to sha1 and sha256. Refer to Intel® QuickAssist Technology Software for Linux* Release Notes (Table 1) for any limitations on using other hash algorithms in the current release.

*注意:哈希算法不限于 sha1 和 sha256。请参阅面向Linux*的英特尔快速辅助技术软件发行说明(*Table 1

Listing 83. Verify the Output of Chained Hash and Stateless Compression

清单 83。验证链式散列和无状态压缩的输出

```
/* Use software to calculate digest and verify digest
*/ if (CPA_STATUS_SUCCESS == status)
{
        PHYS_CONTIG_ALLOC(&pSWDigestBuffer,
        GET_HASH_DIGEST_LENGTH(hashAlg)); status = calSWDigest(sampleData,
                                    bufferSize,
                                    pSWDigestBuffe
                                    r,
                                    GET_HASH_DIGEST_LENGTH(hashAlg),
                                    hashAlg);

    if (CPA_STATUS_SUCCESS == status)
        {
            if (memcmp(pDigestBuffer,
                        pSWDigestBuffer,
                        GET_HASH_DIGEST_LENGTH(hashAlg))
                        )
```

```
/* Use software to calculate digest and verify digest */ if
(CPA_STATUS_SUCCESS == status)
{
        PHYS_CONTIG_ALLOC(&pSWDigestBuffer, GET_HASH_DIGEST_LENGTH(hashAlg)); status =
        calSWDigest(sampleData,
                                    bufferSize,
                                    pSWDigestBuffer,
                                    GET_HASH_DIGEST_LENGTH(hashAlg),
                                    hashAlg);

            if (CPA_STATUS_SUCCESS == status)
        {
            if (memcmp(pDigestBuffer,
                        pSWDigestBuffer,
                        GET_HASH_DIGEST_LENGTH(hashAlg)))
            {
                status = CPA_STATUS_FAIL;
        }
        else
        {
            PRINT_DBG("Digest buffer matches expected output\n");
        }
    }

    PHYS_CONTIG_FREE(pSWDigestBuffer);
}

/* Use zlib to decompress and verify integrity */
//<snippet name="software
decompress"> if (CPA_STATUS_SUCCESS
== status)
{
    struct z_stream_s stream =
    {0}; Cpa8U *pDecompBuffer =
    NULL; Cpa8U *pHWCompBuffer =
    NULL; Cpa8U *pSWCompBuffer =
    NULL; Cpa32U bufferLength =
    0;

    status = inflate_init(&stream);
    if (CPA_STATUS_SUCCESS !=
    status)
    {
        PRINT("zlib stream initialize failed");
    }

    bufferLength = pBufferListSrc->numBuffers *
                   pBufferListSrc->pBuffers->dataLenInBytes;

    if (CPA_STATUS_SUCCESS == status)
    {
        status = PHYS_CONTIG_ALLOC(&pDecompBuffer, bufferLength);
    }

    if (CPA_STATUS_SUCCESS == status)
    {
        status = PHYS_CONTIG_ALLOC(&pHWCompBuffer, bufferLength);
    }

    if (CPA_STATUS_SUCCESS == status)
    {
        status = PHYS_CONTIG_ALLOC(&pSWCompBuffer, bufferLength);
```

```
            }
            else
            {
                PRINT_DBG("Digest buffer matches expected output\n");
            }
        }

        PHYS_CONTIG_FREE(pSWDigestBuffer);
    }

    /* Use zlib to decompress and verify integrity */
    //<snippet name="software decompress"> if
    (CPA_STATUS_SUCCESS == status)
    {
        struct z_stream_s stream = {0};
        Cpa8U *pDecompBuffer = NULL; Cpa8U
        *pHWCompBuffer = NULL; Cpa8U
        *pSWCompBuffer = NULL; Cpa32U
        bufferLength = 0;

        status = inflate_init(&stream); if
        (CPA_STATUS_SUCCESS != status)
        {
            PRINT("zlib stream initialize failed");
        }

        bufferLength = pBufferListSrc->numBuffers *
                          pBufferListSrc->pBuffers->dataLenInBytes;

        if (CPA_STATUS_SUCCESS == status)
        {
            status = PHYS_CONTIG_ALLOC(&pDecompBuffer, bufferLength);
        }

        if (CPA_STATUS_SUCCESS == status)
        {
            status = PHYS_CONTIG_ALLOC(&pHWCompBuffer, bufferLength);
        }

        if (CPA_STATUS_SUCCESS == status)
        {
            status = PHYS_CONTIG_ALLOC(&pSWCompBuffer, bufferLength);
        }

        if (CPA_STATUS_SUCCESS == status)
            copyMultiFlatBufferToBuffer(pBufferListDst, pHWCompBuffer);
        }

        if (CPA_STATUS_SUCCESS == status)
        {
            status = inflate_decompress(&stream,
                                        pHWCompBuffe
                                        r,
                                        bufferLength
                                        ,
                                        pDecompBuffe
                                        r,
                                        bufferLength
                                        );
           if (CPA_STATUS_SUCCESS != status)
            {
                PRINT_ERR("Decompress data on zlib stream failed\n");
            }
        }

        if (CPA_STATUS_SUCCESS == status)
        {
            /* Compare with original Src buffer */
            if (memcmp(pDecompBuffer, sampleData, bufferSize))
            {
               status = CPA_STATUS_FAIL;
                PRINT_ERR("Decompression does not match source buffer\n");
            }
            else
            {
                PRINT_DBG("Decompression matches source buffer\n");
            }
        }

        inflate_destroy(&stream);
```

```
        copyMultiFlatBufferToBuffer(pBufferListDst, pHWCompBuffer);
    }

    if (CPA_STATUS_SUCCESS == status)
    {
        status = inflate_decompress(&stream,
                                    pHWCompBuffer,
                                    bufferLength,
                                    pDecompBuffer,
                                    bufferLength);
          if (CPA_STATUS_SUCCESS != status)
        {
            PRINT_ERR("Decompress data on zlib stream failed\n");
        }
    }

    if (CPA_STATUS_SUCCESS == status)
    {
        /* Compare with original Src buffer */
        if (memcmp(pDecompBuffer, sampleData, bufferSize))
        {
              status = CPA_STATUS_FAIL;
            PRINT_ERR("Decompression does not match source buffer\n");
        }
        else
        {
            PRINT_DBG("Decompression matches source buffer\n");
        }
    }

    inflate_destroy(&stream);

    PHYS_CONTIG_FREE(pSWCompBuffer);
    PHYS_CONTIG_FREE(pHWCompBuffer);
    PHYS_CONTIG_FREE(pDecompBuffer);
```

Listing 84. Remove Chained Hash and Stateless Compression Session

清单 84。移除链接的哈希和无状态压缩会话

```
    status = cpaDcChainRemoveSession(dcInstHandle, sessionHdl);
```

```
status = cpadchainrovesession(dcInstHandle, session HDL);
```

§