



Windows* Getting Started Guide

Intel® QuickAssist Technology

Hardware Version 2.x

Production Release

May 2026

Contents

1	Introduction	2
1.1	About This Document	2
1.2	Features Implemented	2
1.3	Package Release Structure	2
1.4	System Configuration	3
1.4.1	Configuring BIOS	3
1.4.2	Configuring Windows*	4
2	Software Installation	5
2.1	Unpacking the Driver Package	5
2.2	Installation Overview	5
2.3	Configure Acceleration Software	6
2.4	Configure Rate Limiting	6
2.5	Uninstall Driver Package	6
3	Compression Test Application	8
3.1	Parcomp Command-Line Options	8
3.2	Parcomp Examples	10
3.2.1	Compress and Decompress via ‘qat’	10
3.2.2	Compress/Decompress Using Gzip Extended Header	11
3.2.3	Compress/Decompress Using Xpress*	12
3.2.4	Compress/Decompress Performance	13
4	Cryptography Test Application	14
4.1	Cngtest Command-Line Options	14
4.2	Cngtest Examples	18
4.2.1	RSA 3072-bit Key	18
4.2.2	ECDH using Curve25519	18
4.2.3	ECDSA using NistP521	19
4.2.4	RSA 2048-bit Key Performance	19



Intel® QuickAssist Getting Started Guide for Windows*

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com](https://www.intel.com).

Intel technologies may require enabled hardware, specific software, or services activation. Check with your system manufacturer or retailer.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548- 4725 or visit www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2024, Intel Corporation. All rights reserved.

Chapter 1

Introduction

1.1 About This Document

This Getting Started Guide documents the instructions to obtain, install, and exercise the Intel® QuickAssist Technology (Intel® QAT) Windows* software for the Hardware Version 2.x package.

In this document, for convenience:

- Software package is used as a generic term for the Intel® QAT Software Package.
- Acceleration driver is used as a generic term for the software that allows the Intel® QAT Software Library APIs to access the Intel® QAT Endpoint(s).

Note: Please refer to the Windows* QAT Release Notes for a list of validated devices/platforms and Operating System versions.

1.2 Features Implemented

Implemented features are listed in Windows* QAT Release Notes or Windows* QAT Technical Guide.

1.3 Package Release Structure

After unpacking the zip file, the directory should contain the following:

Table 1.1:: Windows* Package Release Structure

Files/Directory	Comments
QATx.x.<version>.zip	Top-level Intel® QAT package
./filelist	List of files in this package
./versionfile	The version of this package; this should match the package version
./QuickAssist	Primary folder containing the files necessary for installation, development, license files, and the README.txt
./QuickAssist/Compression	The Windows* QATzip files necessary for data compression development, the compression sample app (parcomp.exe), and CfQat compression driver
./QuickAssist/Crypto	The binaries necessary to register the QAT device for CNG, CPMProvUser driver, the crypto sample app (cngtest.exe), and crypto service
./QuickAssist/Driver	The Windows* QAT base driver, counter manifest file, and firmware files
./QuickAssist/Setup	The Windows* QAT driver installation binary (QatSetup.exe) and MSI package (IntelQAT.msi)

1.4 System Configuration

This section describes the process of configuring the system prior to the Intel® QuickAssist Technology (Intel® QAT) driver installation.

Important: Make sure that the platform and/or CPU SKU has the QAT hardware.

1.4.1 Configuring BIOS

For Intel® Xeon® 4th-, 5th-Generation, and Intel® Xeon® 6 Processors with Intel® QAT Gen4 and Gen4m, depending on the specific SKU, there can be up to 4 QAT endpoints per socket. It may be possible to disable individual QAT endpoints at the BIOS level by following the instructions below:

Important: The BIOS configuration may differ depending on the platform used. The below example is from an Intel® reference platform.

1. Enter BIOS setup.
2. Navigate to the following path where <n> corresponds to the socket containing the QAT endpoint(s) to be disabled:

```
EDKII Menu > Socket Configuration > IIO Configuration > IOAT Configuration >
Sck<n> > IOAT Configuration
```

3. Update the CPM value to Disable for each QAT endpoint to be disabled for each socket.
4. Save changes.
5. Reboot the system.



1.4.2 Configuring Windows*

For the supported Windows* Operating Systems, there should be no additional steps or packages required to use the QAT devices.

Chapter 2

Software Installation

2.1 Unpacking the Driver Package

The driver package is in a zip package. By default, all supported Windows* Operating Systems can extract this driver package.

Note: To install the QAT drivers, you must have Administrator privileges.

2.2 Installation Overview

The installer requires the Microsoft* VC Redistributable. If the redistributable is not available, it will automatically be installed. The recommended installation method for Windows* is to use the QatSetup.exe binary. The QatSetup.exe binary can be found in the relative directory mentioned in [Package Release Structure](#).

Note: For other methods of installation of the QAT driver in Windows*, see the Windows* QAT Technical Guide.

For virtualization (SR-IOV) support without QAT Host services, select the option to install as a “virtualization host” in the installation program. A system restart may be required at the end of the installation in order to fully enable virtualization support.

Important: When doing a Plug and Play (PnP) install of the Windows* QAT driver, only the QAT device driver will be functional. To use hardware accelerated Compression and Crypto services, those services need to be manually started.

To verify that the driver is installed properly, check the Device Manager for devices under ‘Security Accelerator’. The following table shows the corresponding QAT device with its respective Device ID.

Table 2.1:: Windows* QAT HW 2.x Device ID

QAT Accelerator	Device ID
Intel® 4xxx Accelerator	4940
Intel® 4xxx Accelerator Virtual Function	4941
Intel® 401xx Accelerator	4942
Intel® 401xx Accelerator Virtual Function	4943
Intel® 402xx Accelerator	4944
Intel® 402xx Accelerator Virtual Function	4945
Intel® 420xx Accelerator	4946
Intel® 420xx Accelerator Virtual Function	4947

2.3 Configure Acceleration Software

For a detailed overview on how to configure the QAT device in Windows*, see the Windows* QAT Technical Guide.

2.4 Configure Rate Limiting

Rate limiting is a feature designed in the Intel® QuickAssist Technology accelerator software to enforce Service Level Agreements (SLA), which allocate a specified amount of acceleration capacity for a specified service, including symmetric cryptography (SYM), public key encryption (ASYM), and data compression (DC), at a ring-pair or queue-pair (QP) granularity.

The rate limiting solution provides the following capabilities:

- Virtualization technology agnostic SLA management API
- Ability to query rate limiting information for QAT instances in the Guest/Host
- Ability to query rate limiting information for QAT devices in the Host
- Ability to configure rate limiting SLA for service instances based on QP

To use the Rate Limiting feature, a configuration tool is required to set or remove limits on the virtual function devices for the listed services. Please contact your Intel representative to obtain further information regarding this feature.

2.5 Uninstall Driver Package

From the Windows* Desktop, the QAT driver can be uninstalled from the Control Panel via the “Uninstall a Program” submenu. Select the entry with Name “Intel(R) QuickAssist Technology”.

Reboot if necessary.

Important: When doing a driver uninstall using the Windows Device Manager sequentially, the QAT device driver may not be properly removed when multiple QAT devices are present.

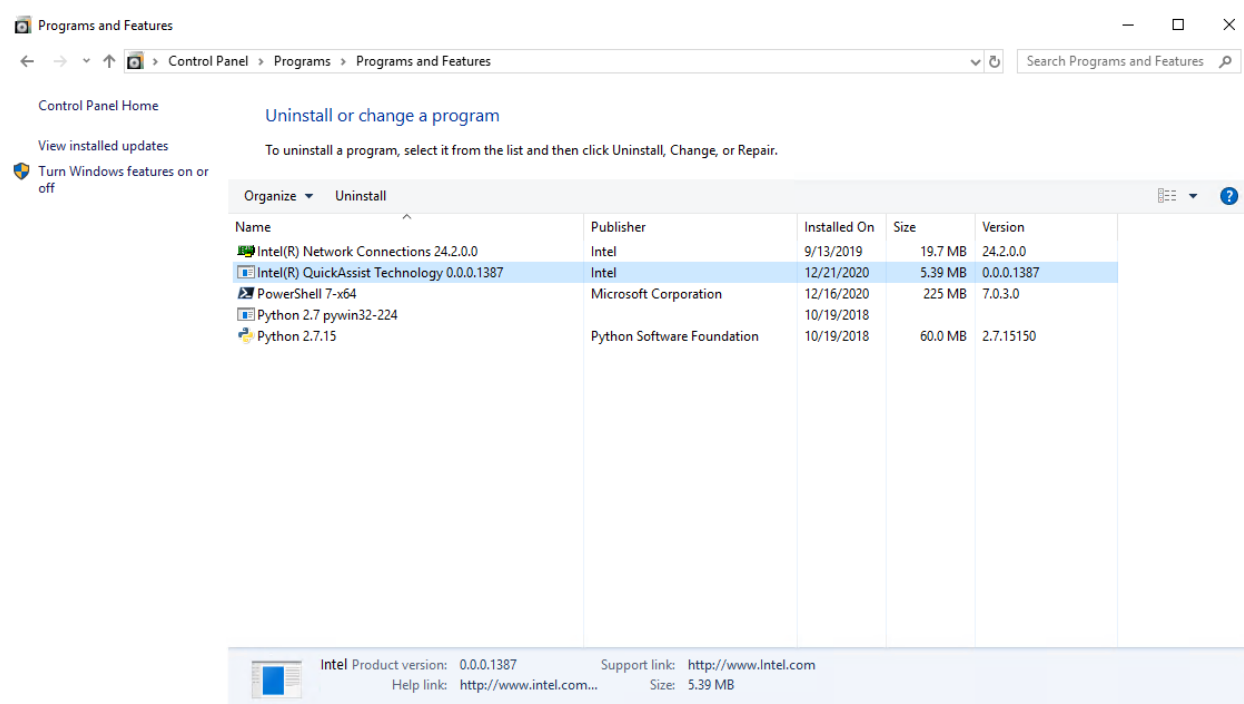


Figure 2.1:: Windows* QAT Uninstallation

Chapter 3

Compression Test Application

This package comes with a tool called 'parcomp' to test the performance of the Intel® QuickAssist Technology accelerator. Parcomp has been built using the QATzip API and library found within the same driver package. The parcomp binary can be found in the following folder upon completion of the installation:

Program Files\Intel\Intel(R) QuickAssist Technology\Compression

Parcomp can be used to measure and report the rate at which compression and decompression operations are performed using the accelerator as well as those operations performed by the default services offered by the system OS.

Note: It is recommended to use parcomp as a privileged user if compressing target files that have elevated privileges. The throughput reported by parcomp is measured around the QATzip API call and does not include the file read and write to disk operations.

The parcomp application needs sufficient system memory to perform the compression and decompression operations. The amount of memory required is dependent on the file size, chunk size, and thread count used for the operation.

3.1 Parcomp Command-Line Options

To see the list of supported command-line options, run either of the following commands:

```
>> parcomp.exe  
>> parcomp.exe -h
```

The output will be as follows:

```
Parcomp: Tool to test compression & decompression  
(c) 2020, Intel(R) Corporation  
  
*Other names and brands may be claimed as the property of others  
  
Usage: -i <srcFilename> -o <dstFilename> [options]  
  
Required options:
```

(continues on next page)

```
-i srcFilename      Input (source) filename
-o dstFilename      Output (destination) filename
```

Optional options can be:

```
-b [cold|warm]      Use cold buffer or not.
-p providerName      Specifies the provider (implementation).
                     Options include:
                     'qat' - QuickAssist accelerated DEFLATE* algorithm
                           with 4 byte headers.
                     'qatzlib' - QuickAssist accelerated DEFLATE*
                           algorithm with zlib* header.
                     'qatms' - Deprecated QuickAssist accelerated
                           DEFLATE* algorithm with MSZIP* header.
                           Only decompression supported.
                     'qatgzip' - QuickAssist accelerated DEFLATE*
                           algorithm with Gzip* header.
                           Note: With qatgzip provider, options
                           -k -t -Q cannot be used in
                           decompression direction.
                           The operation will be executed in a
                           single thread with SW implementation.
                           Parcomp application cannot process
                           Gzip* source buffers bigger than 999MB
                           and the number of iterations is 1.
                     'qatgzipext' - QuickAssist accelerated DEFLATE*
                           algorithm with Gzip* extended header.
                     'xpress' - Software based Xpress* algorithm.
                     'igzip' - Software based DEFLATE* algorithm using
                           igzip DLL.
                     'qatlz4' - QuickAssist accelerated algorithm
                           with lz4* header.

-c chunkSizeInKB     Chunk size, in KB. Default is 64.
                     Files will be divided into chunks of this size
                     (the last chunk may be smaller), and each chunk
                     is compressed separately. State is not maintained
                     between chunks. The chunk size used for decompression
                     must match the value used for compression

-crc bitWidth         CRC bitwidth. Valid value is 64.
-pcrc index           CRC64 configuration to use. Valid values are 0 (ECMA-182) or 1_
↳ (Rocksoft). Default is 0.
-l compressionLevel   Compression level. Default is 1.
                     compressionLevel can range from 1 - 9. Lower values
                     imply less compressibility in less time.
-d                   Decompress the input file. Default is to compress.
-x numLines           Print a summary of the inputs and outputs in a
                     comma-separated variable (CSV) format for easy
                     importing into a spreadsheet. Specify numLines as 1
                     for data only, or 2 to also include a header summary
-t numThreads         Creates specified number of threads, splits input
                     file into numThreads (near-)equal chunks, and
                     performs the operation in each thread. If specifying
```

(continues on next page)

	multiple threads, -Q is also required.
-f cpuFreqInMHz	Specifies the CPU frequency in MHz. If not specified, this will be measured (takes approx. 1 second)
-n numIterations	Specifies the number of iterations (allows you to run the same operation numIterations times) Default is 1.
-Q	Test independent threads writing one process using one session. Uses multiple copies of same input file, outputs one output file per thread.
-k blockSizeInKB	Must be used with -t and threadcount of 1 or more. Separate the source data into several blocks of size specified by blockSizeInKB. -k uses -Q by default.
-h	Print this help message.
-TI	Print individual thread information.
	It supports a maximum of 64 threads (-t) and 20 iterations (-n). (If -t and -n are not specified, default used
↪ is 1)	
The following are applicable for the providers qat (all options) and qatlz4 (-FB, -SW	
↪ and -FT) only:	
-j maxOutstandingJobs	Maximum number of outstanding jobs (requests) that may be outstanding at any one time. Default is 30.
-s	Static compression. Default is dynamic.
-D	Dynamic compression. This is default.
-FB	Enable igzip and LZ4 software fallback.
-SW	Set software only mode. All operations will be processed in
↪ software.	
	Warning: setting both -SW and -FB will result in qatzip
↪ returning an error.	
-FT threshSizeInKB	Threshold value for fallback. If the offload size is less than the threshold, software provider is used.

3.2 Parcomp Examples

Below are some examples of the results obtained by running Parcomp for compression and decompression.

Note: These examples are for illustrative purposes only.

3.2.1 Compress and Decompress via ‘qat’

The following example will compress and decompress a file with the ‘qat’ provider, which is equivalent of using the Deflate algorithm with data format QZ_DEFLATE_4B. The default compression level is 1 and the default chunk size (QATzip hw_buff_sz) is 64KiB.

```
>> parcomp.exe -p qat -i largertext -o largertext.compressed
```

```
Warning: The hw_buff_sz parameter value used for decompression must match the value used
↪ for compression.
```

(continues on next page)

```

Default hw_buff_sz value: 65536 bytes.
hw_buff_sz value used in current execution: 65536 bytes.
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\CompressionFiles\largetext (1000000000 Bytes)
Writing output file: C:\CompressionFiles\largetext.compressed (489318805 Bytes)

Deflation Ratio (%age)      : 48.9
Thruput (uncompressed Mbps): 12312.576
Time (ms)                   : 571.054

Note:- All times exclude file I/O and are measured around the call to the qzCompress()
      ↪ API only.

>> parcomp.exe -p qat -d -i largetext.compressed -o largetext.original

Warning: The hw_buff_sz parameter value used for decompression must match the value used
      ↪ for compression.
Default hw_buff_sz value: 65536 bytes.
hw_buff_sz value used in current execution: 65536 bytes.
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\CompressionFiles\largetext.compressed (489318806 Bytes)
Writing output file: C:\CompressionFiles\largetext.original (1000000000 Bytes)

Inflation Ratio (%age)      : 204.4
Thruput (uncompressed Mbps): 23407.884
Time (ms)                   : 280.976

Note:- All times exclude file I/O and are measured around the call to the qzDecompress()
      ↪ API only.

```

3.2.2 Compress/Decompress Using Gzip Extended Header

The following example will compress and decompress a file with the 'qatgzipext' provider, which is equivalent of using the Deflate algorithm with data format QZ_DEFLATE_GZIP_EXT. The default compression level is 1 and the default chunk size (QATzip hw_buff_sz) is 64KiB.

```

>> parcomp.exe -p qatgzipext -i largetext -o largetext.compressed

Warning: The hw_buff_sz parameter value used for decompression must match the value used
      ↪ for compression.
Default hw_buff_sz value: 65536 bytes.
hw_buff_sz value used in current execution: 65536 bytes.
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: largetext.txt (1000000000 Bytes)
Writing output file: largetext.compressed (396497299 Bytes)

```

(continues on next page)

```

Deflation Ratio (%age)      : 39.6
Thruput (uncompressed Mbps): 14677.658
Time (ms)                   : 356.475

Note:- All times exclude file I/O and are measured around the call to the qzCompress()
↳API only.

>> parcomp.exe -p qatgzipext -d -i targettext.compressed -o targettext.original

Warning: The hw_buff_sz parameter value used for decompression must match the value used
↳for compression.
Default hw_buff_sz value: 65536 bytes.
hw_buff_sz value used in current execution: 65536 bytes.
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: targettext.compressed (396497299 Bytes)
Writing output file: targettext.original (10000000000 Bytes)

Inflation Ratio (%age)      : 252.2
Thruput (uncompressed Mbps): 24304.489
Time (ms)                   : 188.323

Note:- All times exclude file I/O and are measured around the call to the qzDecompress()
↳API only.

```

3.2.3 Compress/Decompress Using Xpress*

The following example will compress and decompress a file with the 'xpress' provider, which is a software based Microsoft compression algorithm.

```

>> parcomp.exe -p xpress -i targettext -o targettext.compressed
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\CompressionFiles\targettext (10000000000 Bytes)
Writing output file: C:\CompressionFiles\targettext.compressed (573799425 Bytes)

Deflation Ratio (%age)      : 57.4
Thruput (uncompressed Mbps): 1070.143
Time (ms)                   : 7283.892

Note:- All times exclude file I/O and are measured around the call to the qzCompress()
↳API only.

>> parcomp.exe -p xpress -d -i targettext.compressed -o targettext.original
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: C:\CompressionFiles\targettext.compressed (573799425 Bytes)

```

(continues on next page)

```
Writing output file: C:\CompressionFiles\largetext.original (10000000000 Bytes)

Inflation Ratio (%age)      : 174.3
Thruput (uncompressed Mbps): 3129.744
Time (ms)                   : 2408.823

Note:- All times exclude file I/O and are measured around the call to the qzDecompress()
↪API only.
```

3.2.4 Compress/Decompress Performance

To achieve optimal performance with QATzip via parcomp, it is recommended to use multiple independent threads for compression and/or decompression. This can be achieved using *-t* (multiple threads) and *-Q* (independent threads).

Note: When using *-Q* as a parameter in the compression command, this will produce a number of identical output files with an appended numeric feather starting with 0. The use of *-k* implicitly brings the same enablement of the *-Q* option.

Unless the file is renamed, this feather will also be required for the decompression command. See full example below.

Important: These performance numbers are samples and are dependent on the platform configuration.

Using large file sizes with high iteration and thread counts will take a large amount of system memory and time to complete, depending on the platform configuration.

```
>> parcomp.exe -i calgary -o calgary.compressed -p qatgzipext -j 30 -Q -t 16 -n 4000 -l
↪1 -c 64 -D
Parcomp: Tool to test compression & decompression
(c) 2020, Intel(R) Corporation

Reading input file: D:\CompressionFiles\calgary (3251493 Bytes)
Warning: No Block size specified, using block size 3251493 bytes.
All threads completed as Expected.

Deflation Ratio (%age)      : 35.9
Thruput (uncompressed Mbps): 312145.964
Processing Block size       : 3175 KB
Block count                 : 1
Time/block (ms)             : 1.333

Note:- All times exclude file I/O and are measured around the call to the
qzCompressExt() API only.
```

Chapter 4

Cryptography Test Application

This package comes with a tool called 'cngtest' to test the performance of the Intel® QuickAssist Technology accelerator. Cngtest has been built with the Microsoft CNG framework found in the Cryptography API: Next Generation Software Development Kit. The cngtest binary can be found in the following folder upon completion of the installation:

Program Files\Intel\Intel(R) QuickAssist Technology\Crypto\Samples\bin

Cngtest can be used to measure and report the rate at which encrypt, decrypt, signhash, verifysignature, finalizekeypair, and secretagreement operations are performed using the Microsoft* CNG framework, with either the default software provider or the provider based on Intel® QuickAssist Technology.

Note: It is recommended to run cngtest as a privileged user.

Unlike parcomp, cngtest does not take any input files. It allocates system memory internally to perform the cryptographic operations.

Kernel-mode tests are no longer supported. All cngtest operations run in user mode.

4.1 Cngtest Command-Line Options

To see the list of supported command-line options, run the following command:

```
>> cngtest.exe -help
```

The output will be as follows:

```
cngtest is a "microbenchmark" which measures and reports the rate (measured
in ops/second) at which encrypt and decrypt operations are performed
using the CNG framework, with one of two providers: the default
(software) provider, or a provider based on Intel(R) QuickAssist
Technology (this requires the presence on the platform of a hardware
accelerator).
```

```
Usage: cngtest [flags]
```

```
The flags are as follows:
```

(continues on next page)

```
-provider={sw|qa}
```

Specifies the provider to use. The default value is "sw", meaning use the default (software) provider. The only other legal value is "qa", which means to use the QuickAssist provider.

Note that the remaining parameters have different defaults depending on the provider, as indicated below:

```
-algo=<algo>
```

Specifies the algorithm to test. The values of <algo> supported are:

rsa: This measures the rate at which RSA decrypt operations (using CRT), and encrypt operations, are performed. Additional flags that can be provided in this case include:

```
-keyLength=<num>
```

Specifies the key size (modulus size) for the operation. The default is 2048. Other legal values include 512, 1024, 1536, 3072 and 4096. Any other value will result in the default 2048 value being used.

ecdsa: This measures the performance of ECDSA algorithm. Additional flag that can be provided in this case include.

```
-ecccurve=<curvename>
```

Specifies the ECC curve name used for ECDSA and ECDH algorithm, default curve name is nistP256, other legal values can refer to CNG Named Elliptic Curves on MSDN

ecdh: This measures the performance of ECDH algorithm. Additional flag that can be provided in this case include.

```
-ecccurve=<curvename>
```

Specifies the ECC curve name used for ECDSA and ECDH algorithm, default curve name is nistP256, other legal values can refer to CNG Named Elliptic Curves on MSDN

dsa: This measures the performance of DSA algorithm (DSA algorithm is not supported in kernel mode, if kernel mode is specified, will route to running user mode test). Additional flags that can be provided in this case include:

```
-keyLength=<num>
```

Specifies the key size (modulus size) for the operation. The default is 2048. Other legal values include 1024 and 3072. Any other value will result in the default 2048 value being used.

dh: This measures the performance of DH algorithm. Additional flags that can be provided in this case include:

```
-keyLength=<num>
```

Specifies the key size (modulus size) for the operation. The default is 2048. Other legal values include 768, 1024, 1536, 3072 and 4096. Any other value will result in the default 2048 value being used.

(continues on next page)

ECDHE-P256-RSA-2K: This measure the performance of RSA decrypt and ECDH P256 secret agreement meanwhile. Additional flag can reference to rsa and ecdh algo.

ECDHE-ECDSA-P256: This measure the performance of ECDSA sign and ECDH secret agreement meanwhile. Additional flag can reference to ecdh and ecdsa algo.

`-padding=<paddingmode>`

RSA algorithm only, ignored for other algorithms.

pkcs1: PKCS1 padding mode.

oaep : OAEP padding mode.

pss : PSS padding mode.

`-numThreads=<num>`

Specifies the number of software threads to spawn. The default value is 2 (sw) or 150 (qa). Note that the number of outstanding requests required to "max out" the hardware is approximately 150. Note too that specifying numThreads=n is equivalent to specifying -minThreads=n and -maxThreads=n.

`-numIter=<numIterations>`

Specifies the number of iterations to perform. The default value is 10000 (sw) or 100000 (qa).

`-affinityMask=<mask>`

Specifies the CPU/core affinity mask for the threads. The affinity mask is interpreted as a bitmask, with each bit indicating a CPU core to which a thread should be affinitized, where core number c is represented as 2^c . Note that the mask is specified as a hexadecimal number, and must begin with the prefix "0x".

For example, to run the software threads on cores 2 and 3, specify -affinityMask=0x0C (binary 00001100). Software threads are assigned to the cores in a round-robin fashion, with the first software thread being assigned to the lowest numbered core, etc.

`-minThreads=<num>`

Specifies the minimum number of software threads. The benchmark will be performed using each number of software threads from minThreads to maxThreads. In this case, the value of -numThreads is ignored.

`-maxThreads=<num>`

Specifies the maximum number of software threads. The benchmark will be performed using each number of software threads from minThreads to maxThreads. In this case, the value of -numThreads is ignored. If minThreads is larger than maxThreads, minThreads and maxThreads are set as default value 1. The maxThreads limit is 150

(continues on next page)

- interop
Specifies that the software and hardware providers should both be executed exactly once each, and the results compared. This is a purely functional interoperability test. All other parameters are ignored in this case.
- encrypt
Measure performance of only the encryption operation for the specified algorithm test. By default performance is measured over both encryption and decryption operations.
- decrypt
Measure performance of only the decryption operation for the specified algorithm test.
- derivekey
Measure performance of only the derive key operation for the specified algorithm (ECDH or DH) test. This option is not supported when using software provider in kernel mode.
- secretderive
Measure performance of the secretagreement + derive key operation for the specified algorithm (ECDH or DH) test. This option is not supported when using software provider in kernel mode.
- finalizesecret
Measure performance of the key generate key + secretagreement operation for the specified algorithm (ECDH or DH) test.
- finalizekey
Measure performance of only the finalize key operation for the specified algorithm (ECDH or DH) test.
- secretagreement
Measure performance of only the secretagreement operation for the specified algorithm (ECDH or DH) test.
- sign
Measure performance of only the sign hash operation for the specified algorithm (ECDSA or DSA) test.
- verify
Measure performance of only the verify signature operation for the specified algorithm (ECDSA or DSA) test.
- generatekey
Measure performance of generate key. This parameter is ignored if algo=DSA is specified.
- debug
Print debug messages.



4.2 Cngtest Examples

Below are some examples of the results obtained by running cngtest for cryptography.

Note: These examples are for illustrative purposes only.

4.2.1 RSA 3072-bit Key

The following example will use cngtest to run RSA 3072-bit Key for encrypt and decrypt operations.

```
>> cngtest.exe -provider=qa -algo=rsa -keyLength=3072
----- QAT Device: Intel(R) 4xxx Accelerator
----- Number of Devices: 4
----- Number of enabled Devices: 4
----- Max number of threads: 192
```

Running in user space...

```
[RunTest] Running with 2 thread(s).
    Time [ns] : 2891517000
    Number of iterations : 100000
    RSA Encrypt Ops/s : 34583.92
    CPU core utilization percentage : 3%
    CPU overall utilization percentage : 1%
    Time [ns] : 50226472900
    Number of iterations : 100000
    RSA Decrypt Ops/s : 1990.98
    CPU core utilization percentage : 1%
    CPU overall utilization percentage : 0%
```

4.2.2 ECDH using Curve25519

The following example will use cngtest to run ECDH with Curve25519 operations.

```
>> cngtest.exe -provider=qa -algo=ecdh -eccurve=curve25519
----- QAT Device: Intel(R) 4xxx Accelerator
----- Number of Devices: 4
----- Number of enabled Devices: 4
----- Max number of threads: 192
```

Running in user space...

```
[RunTest] Running with 2 thread(s).
    Time [ns] : 3873229000
    Number of iterations : 100000
    ECDH Stage 1 Ops/s : 25818.25
    CPU core utilization percentage : 3%
    CPU overall utilization percentage : 2%
    Time [ns] : 2840583800
    Number of iterations : 100000
    ECDH Stage 2 Ops/s : 35204.03
```

(continues on next page)

```
CPU core utilization percentage : 2%
CPU overall utilization percentage : 2%
```

4.2.3 ECDSA using NistP521

The following example will use cngtest to run ECDSA with NistP521 operations.

```
>> cngtest.exe -provider=qa -algo=ecdsa -ecccurve=nistP521
----- QAT Device: Intel(R) 4xxx Accelerator
----- Number of Devices: 4
----- Number of enabled Devices: 4
----- Max number of threads: 192
```

Running in user space...

```
[RunTest] Running with 2 thread(s).
Time [ns] : 20686033200
Number of iterations : 100000
ECDSA Sign Hash Ops/s : 4834.18
CPU core utilization percentage : 1%
CPU overall utilization percentage : 0%
Time [ns] : 68163263500
Number of iterations : 100000
ECDSA Verify Signature Ops/s : 1467.07
CPU core utilization percentage : 1%
CPU overall utilization percentage : 0%
```

4.2.4 RSA 2048-bit Key Performance

To achieve optimal performance with QAT hardware offloading using the Microsoft* CNG framework, it is recommended to use multiple threads for cryptography operations with a set affinity mask.

Important: These performance numbers are samples and are dependent on the platform configuration.

```
>> cngtest.exe -provider=qa -algo=rsa -keyLength=2048 -numThreads=96 -affinityMask=0xff
----- QAT Device: Intel(R) 4xxx Accelerator
----- Number of Devices: 4
----- Number of enabled Devices: 4
----- Max number of threads: 192
```

Running in user space...

```
[RunTest] Running with 96 thread(s).
Time [ns] : 294058600
Number of iterations : 100000
RSA Encrypt Ops/s : 340068.27
CPU core utilization percentage : 85%
CPU overall utilization percentage : 13%
Time [ns] : 1264662500
Number of iterations : 100000
```

(continues on next page)

(continued from previous page)

RSA Decrypt Ops/s	: 79072.48
CPU core utilization percentage	: 12%
CPU overall utilization percentage	: 3%