Intel® Integrated Performance Primitives Cryptography

Developer Guide

Notices and Disclaimers

Contents

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

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

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.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Introducing Intel® Integrated Performance Primitives Cryptography

Use Intel® Integrated Performance Primitives (Intel® IPP) Cryptography to improve performance of cryptographic operations in multimedia, enterprise data, embedded, communications, and scientific/technical applications. The primitives are a common interface for thousands of commonly used algorithms. Using these primitives enables you to automatically tune your application to many generations of processors without changes in your application.

Intel IPP Cryptography library provides high performance implementations of cryptographic functions for several hardware/instruction set generations. Code written with Intel IPP Cryptography automatically takes advantage of available CPU capabilities. This can provide tremendous development and maintenance savings. You can write programs with one optimized execution path, avoiding the alternative of multiple paths (Intel® Streaming SIMD Extensions 2, Supplemental Streaming SIMD Extensions 3, Intel® Advanced Vector Extensions , etc.) to achieve optimal performance across multiple generations of processors.

The goal of the Intel IPP Cryptography software is to provide algorithmic building blocks with

- a simple "primitive" C interface and data structures to enhance usability and portability
- faster time-to-market
- scalability with Intel® hardware

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Notice revision #20201201

What's New

This Developer Guide documents the Intel® Integrated Performance Primitives (Intel® IPP) Cryptography for Intel® oneAPI Base Toolkit release.

Intel IPP Cryptography library is now available through open source. For details about the open source version, see https://software.intel.com/en-us/forums/intel-integrated-performance-primitives/topic/783205

Documentation for older versions of Intel® Integrated Performance Primitives (Intel® IPP) Cryptography is available for download only. For a list of available documentation downloads by product version, see these pages:

- Download Documentation for Intel Parallel Studio XE
- Download Documentation for Intel System Studio

Getting Help and Support

If you did not register your Intel® software product during installation, please do so now at the Intel® Software Development Products Registration Center. Registration entitles you to free technical support, product updates, and upgrades for the duration of the support term.

For general information about Intel technical support, product updates, FAQs, tips and tricks and other support questions, please visit http://www.intel.com/software/products/support/ and the Intel IPP forum https://community.intel.com/t5/Intel-Integrated-Performance/bd-p/integrated-performance-primitives.

NOTE
If your distributor provides technical support for this product, please contact them rather than Intel.

Notational Conventions

The following font and symbols conventions are used in this document:

Italic	*Italic* is used for emphasis and also indicates document names in body text, for example: see *Intel IPP Cryptography Developer Reference*.
`Monospace lowercase`	Indicates filenames, directory names, and pathnames, for example: `/tools/ia32/perfsys`
`Monospace lowercase mixed with UPPERCASE`	Indicates commands and command-line options, for example: `ps_ippcp.exe -f FIRLMS_32f -r firlms.csv`
`UPPERCASE MONOSPACE`	Indicates system variables, for example: `$PATH`.
`monospace italic`	Indicates a parameter in discussions, such as routine parameters, for example: *`pSrc`*; makefile parameters, for example: *`function_list`*. When enclosed in angle brackets, indicates a placeholder for an identifier, an expression, a string, a symbol, or a value, for example: *`<ipp cryptography directory>`*.
[items]	Square brackets indicate that the items enclosed in brackets are optional.
{ item | item }	Braces indicate that only one of the items listed between braces can be selected. A vertical bar (|) separates the items.

The following notations are used to refer to Intel IPP Cryptography directories:

`<install_dir>`	The installation directory for the larger product that includes Intel IPP Cryptography.
`<ipp cryptography directory>`	The main directory where Intel IPP Cryptography is installed: *`<ipp cryptography directory>`*=*`<install_dir>`*`/ippcp/`. Replace this placeholder with the specific pathname in the configuring, linking, and building instructions.

Getting Started with Intel® Integrated Performance Primitives Cryptography

This chapter helps you start using Intel® Integrated Performance Primitives (Intel® IPP) Cryptography by giving a quick overview of some fundamental concepts and showing how to build an Intel® IPP Cryptography program.

Finding Intel® IPP Cryptography on Your System

Intel® Integrated Performance Primitives (Intel® IPP) Cryptography installs in the subdirectory referred to as *<intel ipp cryptography directory>* inside *<install_dir>*. By default, the *<install_dir>* is:

- On Windows* OS: `C:/Program files (x86)/Intel/oneAPI` (on certain systems, instead of `Program Files (x86)`, the directory name is `Program Files`)
- On Linux* OS:
  - admin:`/opt/intel/oneapi`
  - user:`~/intel/oneapi`
- On macOS*: `/opt/intel/oneapi`

The tables below describe the structure of the high-level directories on:

- Windows* OS
- Linux* OS
- macOS*

Windows* OS:

Directory	Contents
`cmake/`	CMake configuration files to use IPP Cryptography in CMake scripts
`components/`	Intel IPP Cryptography interfaces and example files
`documentation/`	Intel IPP Cryptography documentation
`env`	Batch files to set environmental variables in the user shell
`include`	Header files for the library functions
`lib/ia32`	Single-threaded static libraries for the IA-32 architecture
`lib/intel64`	Single-threaded static libraries for the Intel® 64 architecture
`licensing/`	Intel IPP Cryptography license files
`redist/ia32/`	Single-threaded DLLs for applications running on processors with the IA-32 architecture

Directory	Contents
redist/intel64/	Single-threaded DLLs for applications running on processors with the Intel® 64 architecture
tools/custom_library_tool_python	Command-line and GUI tool for building custom dynamic libraries

Linux* OS:

Directory	Contents
components/	Intel IPP Cryptography interfaces and example files
documentation/	Intel IPP Cryptography documentation
env	Batch files to set environmental variables in the user shell
include	Header files for the library functions
lib/ia32	Single-threaded static libraries for the IA-32 architecture
lib/intel64	Single-threaded static libraries for the Intel® 64 architecture
licensing/	Intel IPP Cryptography license files
modulefiles/	Module files for environment configuration
tools/custom_library_tool_python	Command-line and GUI tool for building custom dynamic libraries

macOS*:

Directory	Contents
components/	Intel IPP Cryptography interfaces and example files
documentation/	Intel IPP Cryptography documentation
env	Batch files to set environmental variables in the user shell
include	Header files for the library functions
lib/intel64	Single-threaded static libraries for the Intel® 64 architecture
licensing/	Intel IPP Cryptography license files
tools/custom_library_tool_python	Command-line and GUI tool for building custom dynamic libraries

See Also

Notational Conventions

Setting Environment Variables

When the installation of Intel IPP Cryptography is complete, set the environment variables in the command shell using one of the script files in the `bin` subdirectory of the Intel IPP Cryptography installation directory:

On Windows* OS:

`vars.bat` for the IA-32 and Intel® 64 architectures.

On Linux* OS and macOS*:

Architecture	Shell	Script File
IA-32 and Intel® 64	C	`vars.csh`
IA-32 and Intel® 64	Bash	`vars.sh`

When using the `vars` script, you need to specify the architecture as a parameter. For example:

- `vars.bat ia32`

  sets the environment for Intel IPP Cryptography to use the IA-32 architecture on Windows* OS.
- `. vars.sh intel64`

  sets the environment for Intel IPP Cryptography to use the Intel® 64 architecture on Linux* OS and macOS*.

The scripts set the following environment variables:

Windows* OS	Linux* OS	macOS*	Purpose
`IPPCRYPTOROOT`	`IPPCRYPTOROOT`	`IPPCRYPTOROOT`	Point to the Intel IPP Cryptography installation directory
`LIB`	n/a	n/a	Add the search path for the Intel IPP Cryptography single-threaded libraries
`PATH`	`LD_LIBRARY_PATH`	`DYLD_LIBRARY_PATH`	Add the search path for the Intel IPP Cryptography single-threaded DLLs
`INCLUDE`	n/a	n/a	Add the search path for the Intel IPP Cryptography header files

Compiler Integration

Intel® C++ Compiler and Microsoft Visual Studio* compilers simplify developing with Intel® IPP Cryptography.

On Windows* OS, a default installation of Intel® Parallel Studio XE Composer Edition and Intel® IPP Cryptography installs integration plug-ins. These enable the option to configure your Microsoft Visual Studio* project for automatic linking with Intel IPP Cryptography.

Intel® C++ Compiler also provides command-line parameters to set the link/include directories:

- On Windows* OS:

  `/Qipp:crypto`
- On Linux* OS and macOS*:

  `-ipp:crypto` and `-ipp:nonpic_crypto`

See Also

Linking Your Microsoft* Visual Studio* Project with Intel IPP Cryptography

Linking Your Application with Intel® IPP Cryptography

Building Intel® IPP Cryptography Applications

The code example below represents a short application to help you get started with Intel® IPP Cryptography:

```
#include "ipp.h"
#include <stdio.h>
int main(int argc, char* argv[])
{
   const IppLibraryVersion *lib;
   IppStatus status;
   Ipp64u mask, emask;

   /* Get IPP Cryptography library version info */
   lib = ippcpGetLibVersion();
   printf("%s %s\n", lib->Name, lib->Version);

   /* Get CPU features and features enabled with selected library level */
   status = ippcpGetCpuFeatures( &mask );
   if( ippStsNoErr == status ) {
      emask = ippcpGetEnabledCpuFeatures();
      printf("Features supported by CPU\tby IPP\n");
      printf("---------------------------------------\n");
      printf("  ippCPUID_MMX        = ");
      printf("%c\t%c\t",( mask & ippCPUID_MMX ) ? 'Y':'N',( emask & ippCPUID_MMX ) ? 'Y':'N');
      printf("Intel(R) Architecture MMX technology supported\n");
      printf("  ippCPUID_SSE        = ");
      printf("%c\t%c\t",( mask & ippCPUID_SSE ) ? 'Y':'N',( emask & ippCPUID_SSE ) ? 'Y':'N');
      printf("Intel(R) Streaming SIMD Extensions\n");
      printf("  ippCPUID_SSE2       = ");
      printf("%c\t%c\t",( mask & ippCPUID_SSE2 ) ? 'Y':'N',( emask & ippCPUID_SSE2 ) ? 'Y':'N');
      printf("Intel(R) Streaming SIMD Extensions 2\n");
      printf("  ippCPUID_SSE3       = ");
      printf("%c\t%c\t",( mask & ippCPUID_SSE3 ) ? 'Y':'N',( emask & ippCPUID_SSE3 ) ? 'Y':'N');
      printf("Intel(R) Streaming SIMD Extensions 3\n");
      printf("  ippCPUID_SSSE3      = ");
      printf("%c\t%c\t",( mask & ippCPUID_SSSE3 ) ? 'Y':'N',( emask & ippCPUID_SSSE3 ) ?
'Y':'N');
      printf("Intel(R) Supplemental Streaming SIMD Extensions 3\n");
      printf("  ippCPUID_MOVBE      = ");
      printf("%c\t%c\t",( mask & ippCPUID_MOVBE ) ? 'Y':'N',( emask & ippCPUID_MOVBE ) ?
'Y':'N');
      printf("The processor supports MOVBE instruction\n");
      printf("  ippCPUID_SSE41      = ");
      printf("%c\t%c\t",( mask & ippCPUID_SSE41 ) ? 'Y':'N',( emask & ippCPUID_SSE41 ) ?
'Y':'N');
      printf("Intel(R) Streaming SIMD Extensions 4.1\n");
      printf("  ippCPUID_SSE42      = ");
      printf("%c\t%c\t",( mask & ippCPUID_SSE42 ) ? 'Y':'N',( emask & ippCPUID_SSE42 ) ?
'Y':'N');
      printf("Intel(R) Streaming SIMD Extensions 4.2\n");
      printf("  ippCPUID_AVX        = ");
      printf("%c\t%c\t",( mask & ippCPUID_AVX ) ? 'Y':'N',( emask & ippCPUID_AVX ) ? 'Y':'N');
      printf("Intel(R) Advanced Vector Extensions instruction set\n");
      printf("  ippAVX_ENABLEDBYOS  = ");
      printf("%c\t%c\t",( mask & ippAVX_ENABLEDBYOS ) ? 'Y':'N',( emask & ippAVX_ENABLEDBYOS ) ?
'Y':'N');
      printf("The operating system supports Intel(R) AVX\n");
      printf("  ippCPUID_AES        = ");
```

```
      printf("%c\t%c\t",( mask & ippCPUID_AES ) ? 'Y':'N',( emask & ippCPUID_AES ) ? 'Y':'N');
      printf("Intel(R) AES instruction\n");
      printf("  ippCPUID_SHA        = ");
      printf("%c\t%c\t",( mask & ippCPUID_SHA ) ? 'Y':'N',( emask & ippCPUID_SHA ) ? 'Y':'N');
      printf("Intel(R) SHA new instructions\n");
      printf("  ippCPUID_CLMUL      = ");
      printf("%c\t%c\t",( mask & ippCPUID_CLMUL ) ? 'Y':'N',( emask & ippCPUID_CLMUL ) ?
'Y':'N');
      printf("PCLMULQDQ instruction\n");
      printf("  ippCPUID_RDRAND     = ");
      printf("%c\t%c\t",( mask & ippCPUID_RDRAND ) ? 'Y':'N',( emask & ippCPUID_RDRAND ) ?
'Y':'N');
      printf("Read Random Number instructions\n");
      printf("  ippCPUID_F16C       = ");
      printf("%c\t%c\t",( mask & ippCPUID_F16C ) ? 'Y':'N',( emask & ippCPUID_F16C ) ? 'Y':'N');
      printf("Float16 instructions\n");
      printf("  ippCPUID_AVX2       = ");
      printf("%c\t%c\t",( mask & ippCPUID_AVX2 ) ? 'Y':'N',( emask & ippCPUID_AVX2 ) ? 'Y':'N');
      printf("Intel(R) Advanced Vector Extensions 2 instruction set\n");
      printf("  ippCPUID_AVX512F    = ");
      printf("%c\t%c\t",( mask & ippCPUID_AVX512F ) ? 'Y':'N',( emask & ippCPUID_AVX512F ) ?
'Y':'N');
      printf("Intel(R) Advanced Vector Extensions 3.1 instruction set\n");
      printf("  ippCPUID_AVX512CD   = ");
      printf("%c\t%c\t",( mask & ippCPUID_AVX512CD ) ? 'Y':'N',( emask & ippCPUID_AVX512CD ) ?
'Y':'N');
      printf("Intel(R) Advanced Vector Extensions CD (Conflict Detection) instruction set\n");
      printf("  ippCPUID_AVX512ER   = ");
      printf("%c\t%c\t",( mask & ippCPUID_AVX512ER ) ? 'Y':'N',( emask & ippCPUID_AVX512ER ) ?
'Y':'N');
      printf("Intel(R) Advanced Vector Extensions ER instruction set\n");
      printf("  ippCPUID_ADCOX      = ");
      printf("%c\t%c\t",( mask & ippCPUID_ADCOX ) ? 'Y':'N',( emask & ippCPUID_ADCOX ) ?
'Y':'N');
      printf("ADCX and ADOX instructions\n");
      printf("  ippCPUID_RDSEED     = ");
      printf("%c\t%c\t",( mask & ippCPUID_RDSEED ) ? 'Y':'N',( emask & ippCPUID_RDSEED ) ?
'Y':'N');
      printf("The RDSEED instruction\n");
      printf("  ippCPUID_PREFETCHW  = ");
      printf("%c\t%c\t",( mask & ippCPUID_PREFETCHW ) ? 'Y':'N',( emask & ippCPUID_PREFETCHW ) ?
'Y':'N');
      printf("The PREFETCHW instruction\n");
      printf("  ippCPUID_KNC        = ");
      printf("%c\t%c\t",( mask & ippCPUID_KNC ) ? 'Y':'N',( emask & ippCPUID_KNC ) ? 'Y':'N');
      printf("Intel(R) Xeon Phi(TM) Coprocessor instruction set\n");
   }
   return 0;
}
```

This application consists of three sections:

1. Initialize the Intel IPP Cryptography library. The Intel IPP Cryptography library is auto-initialized with the first call of an Intel IPP Cryptography function.

   In certain debugging scenarios, it is helpful to force a specific implementation layer using `ippcpSetCpuFeatures()`, instead of the best as chosen by the dispatcher.
2. Get the library layer name and version. You can also get the version information using the `ippcpversion.h` file located in the `/include` directory.

3. Show the hardware optimizations used by the selected library layer and supported by CPU.

Building the First Example on Windows* OS

To build the code example above on Windows* OS, follow the steps:

1. Start Microsoft Visual Studio* and create an empty C++ project.
2. Add a new c file and paste the code into it.
3. Set the include directories and the linking model as described in Linking Your Microsoft* Visual Studio* Project with Intel IPP Cryptography.
4. Compile and run the application.

Building the First Example on Linux* OS

To build the code example above on Linux* OS, follow the steps:

1. Paste the code into the editor of your choice.
2. Make sure the compiler and Intel IPP Cryptography variables are set in your shell. For information on how to set environment variables see Setting Environment Variables.
3. Compile with the following command: `icc ippcptest.cpp -o ippcptest -I $IPPCRYPTOROOT/include -L $IPPCRYPTOROOT/lib/<arch> -lippcp`. For more information, see Linking Options.
4. Run the application.

Building the First Example on macOS*

To build the code example above on macOS*, follow the steps:

1. Paste the code into the editor of your choice.
2. Make sure the compiler and Intel IPP Cryptography variables are set in your shell. For information on how to set environment variables see Setting Environment Variables.
3. Compile with the following command: `icc ippcptest.cpp -o ippcptest -I $IPPCRYPTOROOT/include -L $IPPROOT/lib/ -lippcp`. For more information, see Linking Options.
4. Run the application.

See Also

Linking Your Microsoft* Visual Studio* Project with Intel IPP Cryptography
Setting Environment Variables
Linking Options
Dispatching

Finding the Intel® IPP Cryptography Documentation

The *<install_dir>*`/documentation/` directory, set up during installation, includes a lot of helpful documentation related to Intel® IPP Cryptography. See the `get_started.htm` file for a listing of all the available documents with links or pointers to their locations.

The Intel IPP forum and knowledge base can be useful locations to search for questions not answered by the documents above. Please see: http://software.intel.com/en-us/forums/intel-integrated-performance-primitives/ .

See Also

Finding Intel® IPP Cryptography on Your System

Intel® Integrated Performance Primitives Cryptography Theory of Operation

2

This section discusses dispatching of the Intel® Integrated Performance Primitives (Intel® IPP) Cryptography libraries to specific processors, provides functions and parameters naming conventions, and explains the data types on which Intel IPP Cryptography performs operations.

Dispatching

Intel® IPP Cryptography uses multiple function implementations optimized for various CPUs. Dispatching refers to detection of your CPU and selecting the corresponding Intel IPP Cryptography binary path. For example, the `ippcp` library in the `/redist/intel64/ippcp` directory contains cryptographic functions optimized for 64-bit applications on processors with Intel® Advanced Vector Extensions (Intel® AVX) enabled such as the 2nd Generation Intel® Core™ processor family.

A single Intel IPP Cryptography function, for example `ippsSHA256Update()`, may have many versions, each one optimized to run on a specific Intel® processor with specific architecture, for example, the 64-bit version of this function optimized for the 2nd Generation Intel® Core™ processor is `e9_ippsSHA256Update()`, and the version optimized for 64-bit applications on processors with Intel® Streaming SIMD Extensions 4.2 (Intel® SSE 4.2) is `y8_ippsSHA256Update()`. This means that a prefix before the function name determines the CPU model. However, during normal operation the dispatcher determines the best version and you can call a generic function (`ippsSHA256Update()` in this example).

Intel® IPP Cryptography is designed to support application development on various Intel® architectures. This means that the API definition is common for all processors, while the underlying function implementation takes into account the strengths of each hardware generation.

By providing a single cross-architecture API, Intel IPP Cryptography enables you to port features across Intel® processor-based desktop, server, and mobile platforms. You can use your code developed for one processor architecture for many processor generations.

The following table shows processor-specific codes that Intel IPP Cryptography uses:

Description of Codes Associated with Processor-Specific Libraries

IA-32 Intel® architecture	Intel® 64 architecture	Windows*	Linux* OS	Android*	macOS*	Description
w7		+	+			Optimized for processors with Intel SSE2
s8	**n8**	+	+	+		Optimized for processors with Supplemental Streaming SIMD Extensions 3 (SSSE3)
	m7	+	+	+	+	Optimized for processors with Intel SSE3
p8	**y8**	+	+	+	+	Optimized for processors with Intel SSE4.2
g9	**e9**	+	+	+	+	Optimized for processors with Intel® Advanced Vector Extensions (Intel® AVX) and Intel® Advanced Encryption Standard New Instructions (Intel® AES-NI)

IA-32 Intel® architecture	Intel® 64 architecture	Windows*	Linux* OS	Android*	macOS*	Description
h9	**l9**	+	+	+	+	Optimized for processors with Intel® Advanced Vector Extensions 2 (Intel® AVX2)
	k0	+	+		+	Optimized for processors with Intel® Advanced Vector Extensions 512 (Intel® AVX-512)
	n0	+	+			Optimized for processors with Intel® Advanced Vector Extensions 512 (Intel(R) AVX-512) for Intel(R) Many Integrated Core Architecture (Intel(R) MIC Architecture)

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Notice revision #20201201

Function Naming Conventions

Intel IPP Cryptography functions have the same naming conventions for all domains.

Function names in Intel IPP Cryptography have the following general format:

`ipp<data-domain><name>[_<descriptor>](<parameters>)`

> **NOTE**
> The support functions in Intel IPP Cryptography like `ippcpGetCpuFeatures()` do not need an input data type. These functions have `ippcp` as a prefix without the data-domain field.

Data-domain

The *data-domain* element is a single character indicating type of input data. Intel IPP Cryptography supports the following data-domain:

`s`	one-dimensional operations on signals, vectors, buffers

Parameters

The *parameters* element specifies the function parameters (arguments).

The order of parameters is as follows:

- All source operands. Constants follow vectors.
- All destination operands. Constants follow vectors.
- Other, operation-specific parameters.

A parameter name has the following conventions:

- All parameters defined as pointers start with *p*, for example, *pBuffer*, *pSrc*; parameters defined as double pointers start with *pp*, for example, *ppData*. All parameters defined as values start with a lowercase letter, for example, *length*, *bitSize*, *keyLen*.

- Each new part of a parameter name starts with an uppercase character, without underscore; for example, *pSrc*, *bitSize*, *pResult*.
- Each parameter name specifies its functionality. Source parameters are named *pSrc* or *src*, in some cases followed by names or numbers, for example, *pSrc2*, *srcLen*. Output parameters are named *pDst* or *dst* followed by names or numbers, for example, *pDst2*, *dstLen*. For in-place operations, the input/output parameter contains the name *pSrcDst* or *srcDst*.

See Also

Support Functions

Support Functions

There are several general purpose functions that simplify using the library and report information on how it is working:

- `GetCpuFeatures/ SetCpuFeatures/GetEnabledCpuFeatures`
- `GetStatusString`
- `GetLibVersion`

GetCpuFeatures/ SetCpuFeatures/GetEnabledCpuFeatures

In some cases like debugging and performance analysis, you may want to get the data on the difference between various processor-specific codes on the same machine. Use the `ippcpSetCpuFeatures` function for this. This function sets the dispatcher to use the processor-specific code according to the specified set of CPU features. You can obtain features supported by CPU using `ippcpGetCpuFeatures` and obtain features supported by the currently dispatched Intel IPP Cryptography code using `ippcpGetEnabledCpuFeatures`. If you need to enable support of some CPU features without querying the system (without using the CPUID instruction call), you must set the `ippCPUID_NOCHECK` bit for `ippcpSetCpuFeatures`, otherwise, only the features supported by the current CPU are set.

The `ippcpGetCpuFeatures`, `ippcpGetEnabledCpuFeatures`, and `ippcpSetCpuFeatures` functions are a part of the `ippCP` library.

GetStatusString

The `ippcpGetStatusString` function decodes the numeric status return value of Intel® IPP Cryptography functions and converts them to a human-readable text:

```
Ipp64u mask;
status = ippcpGetCpuFeatures(&mask);
if( status != ippStsNoErr ) {
        printf("ippcpGetCpuFeatures() Error:\n");
        printf("%s\n", ippcpGetStatusString(status) );
        return -1;
}
```

The `ippcpGetStatusString` function is a part of the `ippCP` library.

GetLibVersion

The `GetLibVersion` function returns information about the library layer in use from the dispatcher. The code snippet below demonstrates the usage of the `ippcpGetLibVersion`:

```
const IppLibraryVersion* lib = ippcpGetLibVersion();
printf("%s %s %d.%d.%d.%d\n", lib->Name, lib->Version,
       lib->major, lib->minor, lib->majorBuild, lib->build);
```

For more information about the Intel IPP Cryptography functions see the *Developer Reference for Intel® Integrated Performance Primitives Cryptography* available in Intel® Software Documentation Library.

See Also

Intel® Software Documentation Library

Linking Your Application with Intel® Integrated Performance Primitives Cryptography

3

This section discusses linking options available in Intel® Integrated Performance Primitives (Intel® IPP) Cryptography.

The Intel IPP Cryptography library supports the following linking options:

- Single-threaded dynamic
- Single-threaded static
- Multi-threaded dynamic
- Multi-threaded static

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Notice revision #20201201

Linking Options

Intel® Integrated Performance Primitives (Intel® IPP) Cryptography is distributed as:

- **Static library:** static linking results in a standalone executable
- **Dynamic/shared library:** dynamic linking defers function resolution until runtime and requires that you bundle the redistributable libraries with your application

The following table provides description of libraries available for linking.

	Single-threaded
	(non-threaded)
Description	Suitable for application-level threading
Found in	Main package
	After installation: *<ipp cryptography directory>*/lib/*<arch>*
Static linking	Windows* OS: `mt` suffix in a library name (`ippcpmt.lib`)
	Linux* OS and macOS*: no suffix in a library name (`libippcp.a`)
Dynamic Linking	Default (no suffix)
	Windows* OS: `ippcp.lib`
	Linux* OS: `libippcp.a`
	macOS*: `libippcp.dylib`

NOTE
On Linux* OS and macOS* Intel IPP Cryptography library depends on the Intel® C++ Compiler Classic runtime library `libirc.a`. You should add a link to this library into your project. You can find this library in *<ipp cryptography directory>*/lib or *<intel compiler directory>*/lib folders.

See Also

Linking Your Microsoft* Visual Studio* Project with Intel IPP Cryptography

Programming with Intel® Integrated Performance Primitives Cryptography in the Microsoft* Visual Studio* IDE

4

This section provides instructions on how to configure your Microsoft* Visual Studio* IDE to link with the Intel® IPP Cryptography, explains how to access Intel IPP Cryptography documentation and use IntelliSense* Sense features.

Linking Your Microsoft* Visual Studio* Project with Intel IPP Cryptography

To link your Microsoft* Visual Studio* IDE project with Intel IPP Cryptography, follow these steps:

1. Set the Path for the Intel IPP Cryptography header and library files in Microsoft* Visual Studio* IDE:
   a. Right click your project, then select **Properties** > **Configuration Properties** > **VC++ Directories**.
   b. Select the **<Edit...>** option in drop down list of **Include Directories**, click the **New Line** button to add a new line and select the `include` subdirectory of the *<ipp cryptography directory>* folder.
   For example, the default Intel IPP Cryptography header file directory is located at *<ipp cryptography directory>*`\include`.
   c. Select the **<Edit...>** option in drop down list of **Library Directories**, click the **New Line** button to add a new line and select the `lib` subdirectory of the *<ipp cryptography directory>* folder.
   For example, the default Intel IPP Cryptography directory is located at *<ipp cryptography directory>*`\lib\`*<arch>*, where *<arch>* can be `ia32` or `intel64`.
2. Link the Intel IPP Cryptography library into your project:
   a. Right click your project, then select **Properties** > **Configuration Properties** > **Linker** > **Input**.
   b. Select the **<Edit...>** option in drop down list of **Additional Dependencies** and enter the filename of the Intel IPP Cryptography library you wish to use.
   For example, enter ippcp.lib for dynamic linking or ippcpmt.lib for static linking.
3. Add the Intel IPP Cryptography dynamic library to your executable environment (for DLL)
   You can add the path to the Intel IPP Cryptography dynamic-link library to your `PATH` environment variable permanently by following these steps:
   a. Open the **Start** menu and click the **Control Panel** icon in the **Windows System** group.
   b. Select **Large icons** in the **View by:** drop-down list, then find and click **System**.
   c. Click **Advanced system settings** in the column on the left side of the window.
   d. Select the **Advanced** tab and click the **Environment Variables...** button.
   e. Select the **Path** variable in the list of user or system variables and click **Edit...**.
   f. Append *;<ipp cryptography directory>*`\redist\`*<arch>* to the contents of the **Variable value** field and click **OK**.
   Alternatively, to launch the Microsoft* Visual Studio* IDE from a preconfigured command line environment:
   a. Run `{{vars.bat}}` from the directory *<ipp cryptography directory>*`\env\` in a command window.
   b. Start `<Microsoft Visual Studio>\IDE\devenv.exe` in the same command window.

Using the IntelliSense* Features

Intel IPP Cryptography supports two Microsoft* Visual Studio IntelliSense* features that support language references: Complete Word and Parameter Info.

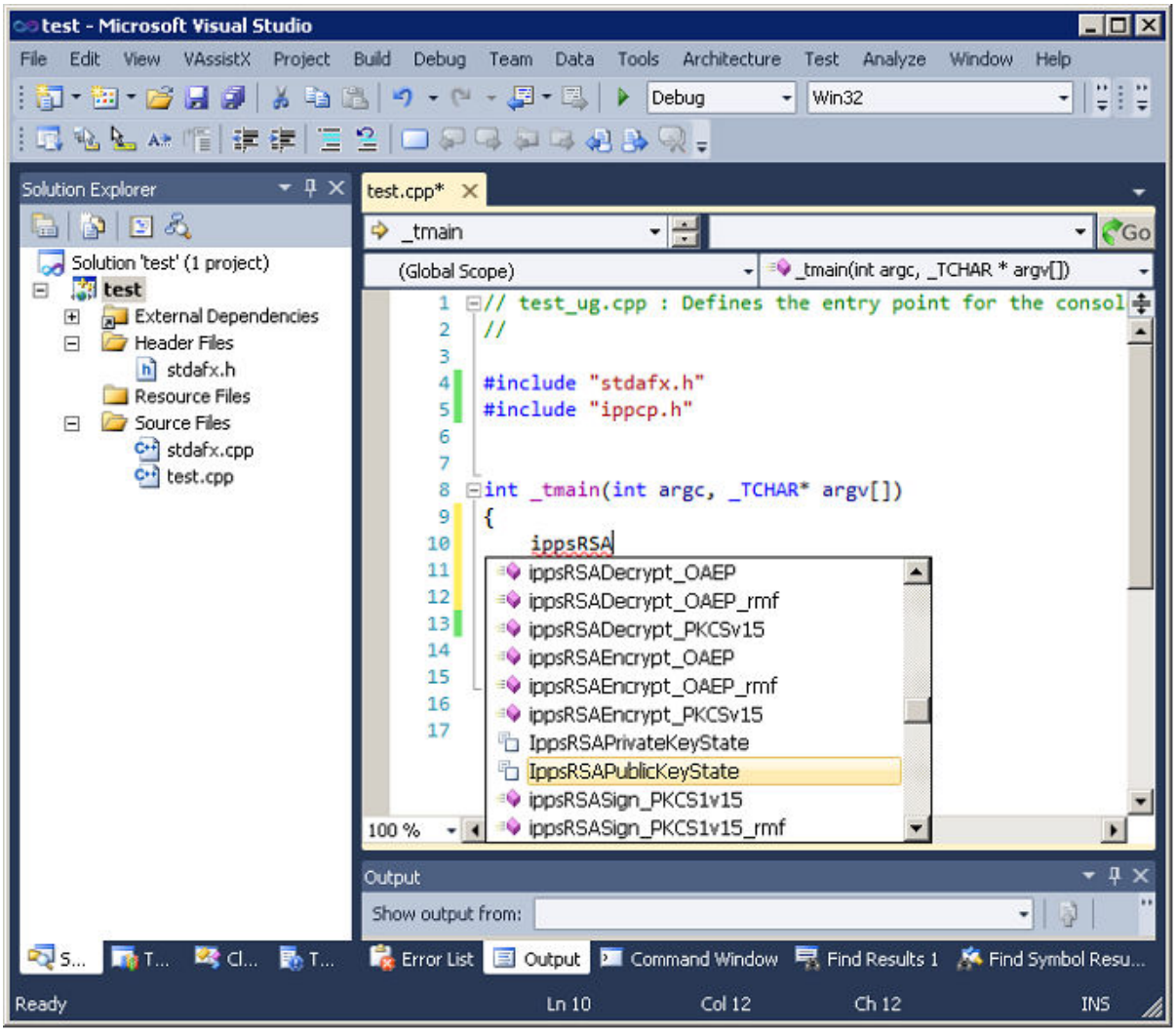
> **NOTE**
> Both features require header files. Therefore, to benefit from IntelliSense, make sure the path to the include files is specified in the Visual Studio solution settings. On how to do this, see Linking Your Microsoft* Visual Studio* Project with Intel IPP Cryptography.

Complete Word

For a software library, the *Complete Word* feature types or prompts for the rest of the name defined in the header file once you type the first few characters of the name in your code.

Provided your C/C++ code contains the include statement with the appropriate Intel IPP Cryptography header file, to complete the name of the function or named constant specified in the header file, follow these steps:

1. Type the first few characters of the name (for example, `ippsRSA`).
2. Press **Alt** + **RIGHT ARROW** or **Ctrl** + **SPACEBAR** If you have typed enough characters to eliminate ambiguity in the name, the rest of the name is typed automatically. Otherwise, the pop-up list of the names specified in the header file opens - see the figure below.



3. Select the name from the list, if needed.

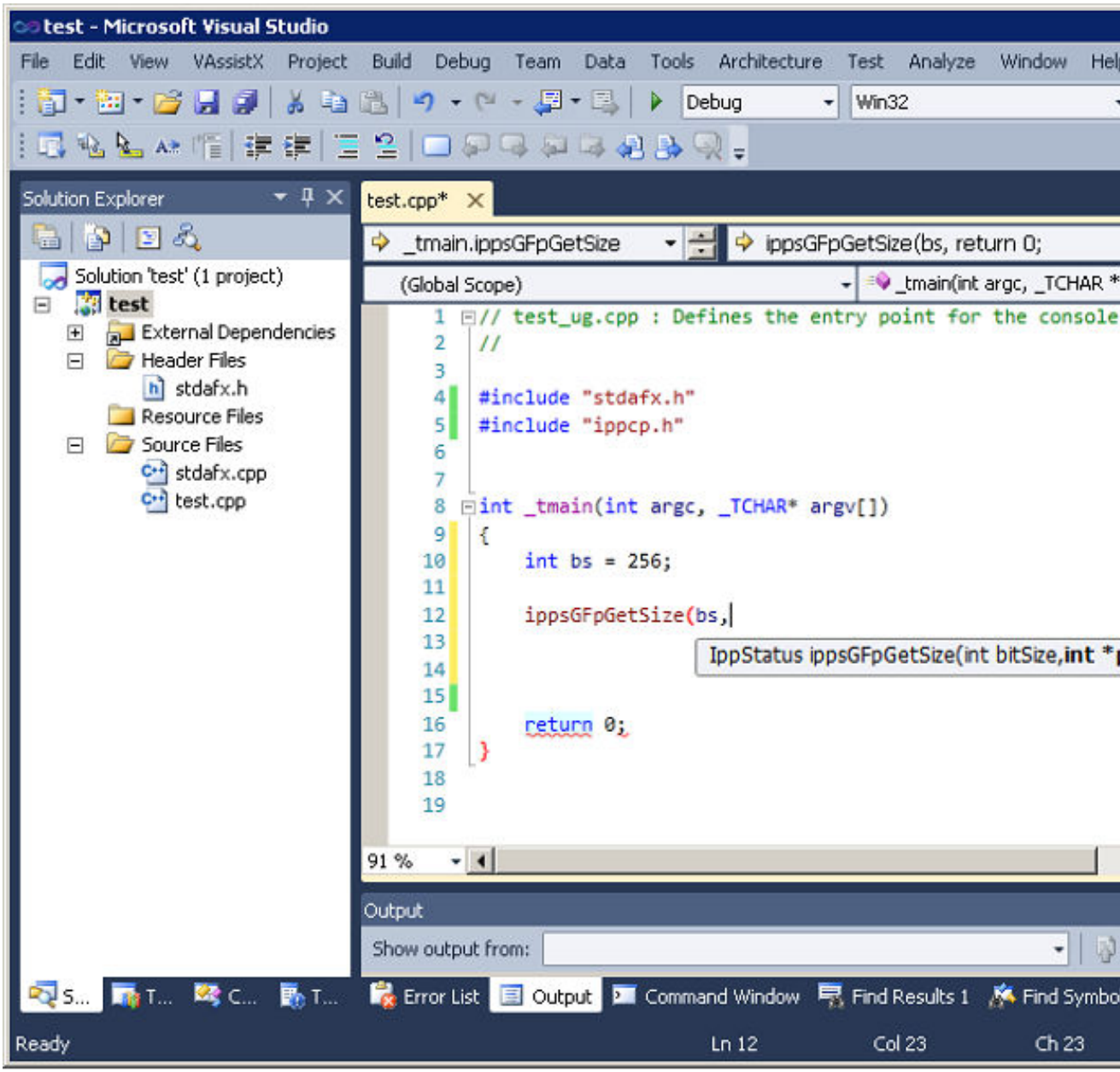
Parameter Info

The *Parameter Info* feature displays the parameter list for a function to give information on the number and types of parameters.

To get the list of parameters of a function specified in the header file, follow these steps:

1. Type the function name
2. Type the opening parenthesis

A tooltip appears with the function API prototype, and the current parameter in the API prototype is highlighted - see the figure below.



See Also

Linking Your Microsoft* Visual Studio* Project with Intel IPP Cryptography

Appendix: Performance Test Tool (perfsys) Command Line Options

Intel® Integrated Performance Primitives (Intel® IPP Cryptography) installation includes a command-line tool for performance testing in the `<install_dir>/tools/perfsys` directory. The `ps_ippcp` executable measures performance for all Intel IPP Cryptography functions. The `ps_crypto_mb` executable measures performance for all Intel IPP Multi-buffer Cryptography functions. Note that `ps_crypto_mb` executable can be ran on 3rd Gen Intel® Xeon® Scalable processors only.

Many factors may affect Intel IPP Cryptography performance. One of the best way to understand them is to run multiple tests in the specific environment you are targeting for optimization. The purpose of the perfsys tools is to simplify performance experiments and empower developers with useful information to get the best performance from Intel IPP Cryptography functions.

With the command-line options you can:

- Create a list of functions to test
- Set parameters for each function
- Set image/buffer sizes

To simplify re-running specific tests, you can define the functions and parameters in the initialization file, or enter them directly from the console.

The command-line format is:

```
ps_ippcp.exe [option_1] [option_2] ... [option_n]
```

To invoke the short reference for the command-line options, use `-?` or `-h` commands:

```
ps_ippcp.exe -h
```

The command-line options are divided into several groups by functionality. You can enter options in arbitrary order with at least one space between each option name. Some options (like `-r`, `-R`, `-o`, `-O`) may be entered several times with different file names, and option `-f` may be entered several times with different function patterns. For detailed descriptions of the perfsys command-line options see the following table:

Performance Test Tool Command Line Options

Group	Option	Description
Set optimization layer to test	`-T[`*`cpu-features`*`]`	Call `ippcpSetCpuFeatures`
Report Configuration	`-A<Timing|Params|Misalign|All>`	Prompt for the parameters before every test from console
	`-o[`*`<file-name>`*`]`	Create *`<file-name>`*`.txt` file and write console output to it
	`-O[`*`<file-name>`*`]`	Add console output to the file *`<file-name>`*`.txt`
	`-L` *`<ERR|WARN|PARM|INFO|TRACE>`*	Set detail level of the console output
	`-r[`*`<file-name>`*`]`	Create *`<file-name>`*`.csv` file and write perfsys results to it
	`-R[`*`<file-name>`*`]`	Add test results to the file *`<file-name>`*`.csv`
	`-q[`*`<file-name>`*`]`	Create *`<file-name>`*`.csv` and write function parameter name lines to it

Group	Option	Description
	-q+	Add function parameter name lines to perfsys results table file
	-Q	Exit after creation of the function parameter name table
	-u[<*file-name*>]	Create <*file-name*>.csv file and write summary table ('_sum' is added to default file name)
	-U[<*file-name*>]	Add summary table to the file <*file-name*>.csv ('_sum' is added to default file name)
	-g[<*file-name*>]	Create signal file at the end of the whole testing
	-l<*dir-name*>	Set default directory for output files
	-k<and|or>	Compose different keys (-f, -t, -m) by logical operation
	-F<*func-name*>	Start testing from function with func-name full name
	-Y<*HIGH/NORMAL*>	Set high or normal process priority (normal is default)
	-H[ONLY]	Add 'Interest' column to .csv file [and run only hot tests]
	-N<*num-threads*>	Call ippcpSetNumThreads(<*num-treads*>)
	-s[-]	Sort or do not sort functions (sort mode is default)
	-e	Enumerate tests and exit
	-v	Display the version number of the perfsys and exit
	-@<*file-name*>	Read command-line options for the specified file
Set function parameters	-d<*name*>=<*value*>	Set perfsys parameter value
Initialization files	-i[<*file-name*>]	Read perfsys parameters from the file <*file-name*>.ini
	-I[<*file-name*>]	Write perfsys parameters to the file <*file-name*>.ini and exit
	-P	Read tested function names from the .ini file
	-n<*title-name*>	Set default title name for .ini file and output files
	-p<*dir-name*>	Set default directory for .ini file and input test data files
Select functions	-f <*or-pattern*>	Run tests for functions with pattern in their names, case sensitive
	-f-<*not-pattern*>	Do not test functions with pattern in their names, case sensitive

Group	Option	Description
	`-f+<and-pattern>`	Run tests only for functions with `pattern` in their names, case sensitive
	`-f=<eq-pattern>`	Run tests for functions with `pattern` full name
	`-t[-|+|=]<pattern>`	Run (do not run) tests with `pattern` in test name
	`-m[-|+|=]<pattern>`	Run (do not run) tests registered in file with `pattern` in file name
Help	`-h`	Display short help and exit
	`-hh`	Display extended help and exit
	`-h<key>`	Display extended help for the key and exit

Appendix: Intel® IPP Cryptography Threading and OpenMP* Support

B

All Intel® Integrated Performance Primitives Cryptography functions are thread-safe. They support multithreading in both dynamic and static libraries and can be used in multi-threaded applications. However, if an application has its own threading model or if other threaded applications are expected to run at the same time on the system, it is strongly recommended to use non-threaded/single-threaded libraries.

Some Intel IPP Cryptography functions contain OpenMP* code, which increases performance on multi-processor and multi-core systems.

To see the list of all threaded APIs, refer to the *ThreadedFunctionsList.txt* file located in the documentation directory of the Intel IPP Cryptography installation.

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Notice revision #20201201

Setting Number of Threads

By default, the number of threads for Intel IPP Cryptography threaded libraries follows the OpenMP* default, which is the number of logical processors visible to the operating system. If the value of the `OMP_NUM_THREADS` environment variable is less than the number of processors, then the number of threads for Intel IPP Cryptography threaded libraries equals the value of the `OMP_NUM_THREADS` environment variable.

To configure the number of threads used by Intel IPP Cryptography internally, at the very beginning of an application call the `ippcpSetNumThreads(n)` function, where `n` is the desired number of threads (1, ...). To disable internal parallelization, call the `ippcpSetNumThreads(1)` function.

Getting Information on Number of Threads

To find the number of threads created by the Intel IPP Cryptography, call the `ippcpGetNumThreads` function.

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Notice revision #20201201

Avoiding Nested Parallelization

Nested parallelization may occur if you use a threaded Intel IPP Cryptography function in a multithreaded application. Nested parallelization may cause performance degradation because of thread oversubscription.

For applications that use OpenMP* threading, nested threading is disabled by default, so this is not an issue.

However, if your application uses threading created by a tool other than OpenMP*, you must disable multi-threading in the threaded Intel IPP Cryptography function to avoid this issue.

Disabling Multi-threading (Recommended)

The best option to disable multi-threading is to link your application with the Intel® IPP Cryptography single-threaded (non-threaded) libraries included in the default package and discontinue use of the separately downloaded multi-threaded versions.

You may also call the `ippcpSetNumThreads` function with parameter `1`, but this method may still incur some OpenMP* overhead.

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Notice revision #20201201