# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notices and Disclaimers</td>
<td>3</td>
</tr>
<tr>
<td>Introducing Intel® Integrated Performance Primitives Cryptography</td>
<td>4</td>
</tr>
<tr>
<td>What's New</td>
<td>5</td>
</tr>
<tr>
<td>Getting Help and Support</td>
<td>6</td>
</tr>
<tr>
<td>Notational Conventions</td>
<td>7</td>
</tr>
<tr>
<td>Chapter 1: Getting Started with Intel® Integrated Performance Primitives Cryptography</td>
<td>8</td>
</tr>
<tr>
<td>Finding Intel® IPP Cryptography on Your System</td>
<td></td>
</tr>
<tr>
<td>Setting Environment Variables</td>
<td>10</td>
</tr>
<tr>
<td>Compiler Integration</td>
<td>10</td>
</tr>
<tr>
<td>Building Intel® IPP Cryptography Applications</td>
<td>11</td>
</tr>
<tr>
<td>Finding the Intel® IPP Cryptography Documentation</td>
<td>14</td>
</tr>
<tr>
<td>Chapter 2: Intel® Integrated Performance Primitives Cryptography Theory of Operation</td>
<td>15</td>
</tr>
<tr>
<td>Dispatching</td>
<td></td>
</tr>
<tr>
<td>Function Naming Conventions</td>
<td>16</td>
</tr>
<tr>
<td>Support Functions</td>
<td>17</td>
</tr>
<tr>
<td>Chapter 3: Linking Your Application with Intel® Integrated Performance Primitives Cryptography</td>
<td>19</td>
</tr>
<tr>
<td>Linking Options</td>
<td></td>
</tr>
<tr>
<td>Chapter 4: Programming with Intel® Integrated Performance Primitives Cryptography in the Microsoft* Visual Studio* IDE</td>
<td>21</td>
</tr>
<tr>
<td>Linking Your Microsoft* Visual Studio* Project with Intel IPP Cryptography</td>
<td></td>
</tr>
<tr>
<td>Using the IntelliSense* Features</td>
<td>22</td>
</tr>
<tr>
<td>Appendix A: Performance Test Tool (perfsys) Command Line Options</td>
<td></td>
</tr>
<tr>
<td>Appendix B: Intel® IPP Cryptography Threading and OpenMP* Support</td>
<td></td>
</tr>
<tr>
<td>Setting Number of Threads</td>
<td>28</td>
</tr>
<tr>
<td>Avoiding Nested Parallelization</td>
<td>29</td>
</tr>
</tbody>
</table>
Notices and Disclaimers

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 product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

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.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors which may cause deviations from published specifications. Current characterized errata are available on request.

Intel, the Intel logo, Intel Atom, Intel Core, Intel Xeon Phi, VTune and Xeon 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.

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.

© Intel Corporation.

This software and the related documents are Intel copyrighted materials, and your use of them is governed by the express license under which they were provided to you (License). Unless the License provides otherwise, you may not use, modify, copy, publish, distribute, disclose or transmit this software or the related documents without Intel's prior written permission.

This software and the related documents are provided as is, with no express or implied warranties, other than those that are expressly stated in the License.
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

Intel IPP Cryptography library is available as part of Intel® Parallel Studio XE and Intel® System Studio. It is also provided as a standalone package under the Community Licensing Program.

Intel IPP Cryptography library is also 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

The majority of Intel IPP Cryptography functions offered in different product suites are the same. But there are several Intel IPP Cryptography libraries or domains that are only included in some product packages. The following table provides a summary of Intel IPP Cryptography functionality for different product suites:

<table>
<thead>
<tr>
<th>Intel IPP Cryptography Functionality</th>
<th>Intel® Parallel Studio XE</th>
<th>Intel® System Studio</th>
<th>Intel® IPP Cryptography Standalone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptography function domains</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Intel® IPP libraries for Android* and Intel® Quark™ platforms</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Intel® IPP libraries for macOS*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
What's New


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
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.


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)/IntelSWTools/compilers_and_libraries_2020.x.xxx/<target_os> (on certain systems, instead of Program Files (x86), the directory name is Program Files)
- On Linux* OS and macOS*: /opt/intel/compilers_and_libraries_2020.x.xxx/<linux|mac>

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

- Windows* OS
- Linux* OS
- macOS*

### Windows* OS:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subdirectories of</strong>&lt;intel ipp cryptography directory&gt;</td>
<td></td>
</tr>
<tr>
<td>bin</td>
<td>Batch files to set environmental variables in the user shell</td>
</tr>
<tr>
<td>include</td>
<td>Header files for the library functions</td>
</tr>
<tr>
<td>lib/ia32</td>
<td>Single-threaded static libraries for the IA-32 architecture</td>
</tr>
<tr>
<td>lib/intel64</td>
<td>Single-threaded static libraries for the Intel® 64 architecture</td>
</tr>
<tr>
<td>lib/&lt;arch&gt;/threaded, where &lt;arch&gt; is one of {ia32, intel64}</td>
<td>Multi-threaded static libraries</td>
</tr>
<tr>
<td>examples</td>
<td>Symbolic link to Intel IPP Cryptography example files</td>
</tr>
<tr>
<td><strong>Subdirectories of</strong>&lt;install_dir&gt;</td>
<td></td>
</tr>
<tr>
<td>redist/ia32/ippcp</td>
<td>Single-threaded DLLs for applications running on processors with the IA-32 architecture</td>
</tr>
<tr>
<td>redist/intel64/ippcp</td>
<td>Single-threaded DLLs for applications running on processors with the Intel® 64 architecture</td>
</tr>
<tr>
<td>redist/&lt;arch&gt;/threaded</td>
<td>Multi-threaded DLLs</td>
</tr>
</tbody>
</table>
**Directory** | **Contents**
--- | ---
`documentation/<locale>/ippcp`, where `<locale>` is one of {en, ja} | Intel IPP Cryptography documentation

### Linux* OS:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>**Subdirectories of **<code>&lt;intel ipp cryptography directory&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>bin</code></td>
<td>Scripts to set environmental variables in the user shell</td>
</tr>
<tr>
<td><code>include</code></td>
<td>Header files for the library functions</td>
</tr>
<tr>
<td><code>lib/ia32</code></td>
<td>Single-threaded static libraries for the IA-32 architecture</td>
</tr>
<tr>
<td><code>lib/intel64</code></td>
<td>Single-threaded static libraries for the Intel® 64 architecture</td>
</tr>
<tr>
<td><code>lib/&lt;arch&gt;/threaded</code></td>
<td>Multi-threaded static libraries</td>
</tr>
<tr>
<td><code>lib/&lt;arch&gt;/nonpic</code></td>
<td>Non-PIC single-threaded static libraries</td>
</tr>
<tr>
<td><code>examples</code></td>
<td>Symbolic link to Intel IPP Cryptography example files</td>
</tr>
<tr>
<td><code>components</code></td>
<td>Intel IPP Cryptography interfaces and example files</td>
</tr>
<tr>
<td><code>tool/&lt;arch&gt;/perfsys</code></td>
<td>Command-line tools for Intel IPP Cryptography performance testing</td>
</tr>
<tr>
<td><code>tool/&lt;arch&gt;/staticlib</code></td>
<td>Header files for redefining Intel IPP Cryptography functions to processor-specific counterparts</td>
</tr>
</tbody>
</table>

### macOS*:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>**Subdirectories of **<code>&lt;intel ipp cryptography directory&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>bin</code></td>
<td>Scripts to set environmental variables in the user shell</td>
</tr>
<tr>
<td><code>include</code></td>
<td>Header files for the library functions</td>
</tr>
<tr>
<td><code>lib</code></td>
<td>Single-threaded static FAT libraries</td>
</tr>
<tr>
<td><code>lib/threaded</code></td>
<td>Multi-threaded static FAT libraries</td>
</tr>
<tr>
<td><code>examples</code></td>
<td>Symbolic link to Intel IPP Cryptography example files</td>
</tr>
<tr>
<td><code>components</code></td>
<td>Intel IPP Cryptography interfaces and example files</td>
</tr>
<tr>
<td><code>tool/&lt;arch&gt;/perfsys</code></td>
<td>Command-line tools for Intel IPP Cryptography performance testing</td>
</tr>
</tbody>
</table>
Subdirectories of `<install_dir>`
documentation/`<locale>/ippcp`, where `<locale>` is one of `{en, ja}`

Intel IPP Cryptography documentation

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:

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

On Linux* OS and macOS*:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Shell</th>
<th>Script File</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA-32 and Intel® 64</td>
<td>C</td>
<td>ippcpvars.csh</td>
</tr>
<tr>
<td>IA-32 and Intel® 64</td>
<td>Bash</td>
<td>ippcpvars.sh</td>
</tr>
</tbody>
</table>

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

- `ippcpvars.bat ia32` sets the environment for Intel IPP Cryptography to use the IA-32 architecture on Windows* OS.
- `. ippcpvars.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:

<table>
<thead>
<tr>
<th>Windows* OS</th>
<th>Linux* OS</th>
<th>macOS*</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPPCRYPTOROOT</td>
<td>IPPCRYPTOROOT</td>
<td>IPPCRYPTOROOT</td>
<td>Point to the Intel IPP Cryptography installation directory</td>
</tr>
<tr>
<td>LIB</td>
<td>n/a</td>
<td>n/a</td>
<td>Add the search path for the Intel IPP Cryptography single-threaded libraries</td>
</tr>
<tr>
<td>PATH</td>
<td>LD_LIBRARY_PATH</td>
<td>DYLD_LIBRARY_PATH</td>
<td>Add the search path for the Intel IPP Cryptography single-threaded DLLs</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>n/a</td>
<td>n/a</td>
<td>Add the search path for the Intel IPP Cryptography header files</td>
</tr>
</tbody>
</table>

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:

```c
#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
", 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	by IPP
");
        printf("-----------------------------------------
");
        printf("  ippCPUID_MMX        = ");
        printf("%c	%c	", ( mask & ippCPUID_MMX ) ? 'Y':'N', ( emask & ippCPUID_MMX ) ? 'Y':'N');
        printf("Intel(R) Architecture MMX technology supported
");
        printf("  ippCPUID_SSE        = ");
        printf("%c	%c	", ( mask & ippCPUID_SSE ) ? 'Y':'N', ( emask & ippCPUID_SSE ) ? 'Y':'N');
        printf("Intel(R) Streaming SIMD Extensions
");
        printf("  ippCPUID_SSE2       = ");
        printf("%c	%c	", ( mask & ippCPUID_SSE2 ) ? 'Y':'N', ( emask & ippCPUID_SSE2 ) ? 'Y':'N');
        printf("Intel(R) Streaming SIMD Extensions 2\n")
        printf("  ippCPUID_SSE3       = ");
        printf("%c	%c	", ( mask & ippCPUID_SSE3 ) ? 'Y':'N', ( emask & ippCPUID_SSE3 ) ? 'Y':'N');
        printf("Intel(R) Streaming SIMD Extensions 3\n")
        printf("  ippCPUID_SSSE3      = ");
        printf("%c	%c	", ( mask & ippCPUID_SSSE3 ) ? 'Y':'N', ( emask & ippCPUID_SSSE3 ) ? 'Y':'N');
        printf("Intel(R) Supplemental Streaming SIMD Extensions 3\n")
        printf("  ippCPUID_MOVBE      = ");
        printf("%c	%c	", ( 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");
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 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/en/ippcp` 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
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:

<table>
<thead>
<tr>
<th>Description of Codes Associated with Processor-Specific Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA-32 Intel® 64 architecture</td>
</tr>
<tr>
<td>w7</td>
</tr>
<tr>
<td>s8</td>
</tr>
<tr>
<td>m7</td>
</tr>
<tr>
<td>p8</td>
</tr>
<tr>
<td>g9</td>
</tr>
</tbody>
</table>


### 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 \textit{p}, for example, \textit{pBuffer, pSrc}; parameters defined as double pointers start with \textit{pp}, for example, \textit{ppData}. All parameters defined as values start with a lowercase letter, for example, \textit{length, bitSize, keyLen}.

• Each new part of a parameter name starts with an uppercase character, without underscore; for example, \textit{pSrc, bitSize, pResult}.

• Each parameter name specifies its functionality. Source parameters are named \textit{pSrc} or \textit{src}, in some cases followed by names or numbers, for example, \textit{pSrc2, srcLen}. Output parameters are named \textit{pDst} or \textit{dst} followed by names or numbers, for example, \textit{pDst2, dstLen}. For in-place operations, the input/output parameter contains the name \textit{pSrcDst} or \textit{srcDst}.

\textbf{See Also}

Support Functions

\textbf{Support Functions}

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

• \texttt{GetCpuFeatures/ SetCpuFeatures/GetEnabledCpuFeatures}
• \texttt{GetStatusString}
• \texttt{GetLibVersion}

\texttt{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 \texttt{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 \texttt{ippcpGetCpuFeatures} and obtain features supported by the currently dispatched Intel IPP Cryptography code using \texttt{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 \texttt{ippCPUID\_NOCHECK} bit for \texttt{ippcpSetCpuFeatures}, otherwise, only the features supported by the current CPU are set.

The \texttt{ippcpGetCpuFeatures}, \texttt{ippcpGetEnabledCpuFeatures}, and \texttt{ippcpSetCpuFeatures} functions are a part of the \texttt{ippCP} library.

\texttt{GetStatusString}

The \texttt{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 \texttt{ippcpGetStatusString} function is a part of the \texttt{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:

```c
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

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

**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

**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.

<table>
<thead>
<tr>
<th>Linking</th>
<th>Single-threaded</th>
<th>Multi-threaded</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>(non-threaded)</td>
<td>(internally threaded)</td>
</tr>
<tr>
<td>Suitable for application-level threading</td>
<td>Use only when no other form of threading is active</td>
<td></td>
</tr>
<tr>
<td><strong>Found in</strong></td>
<td>Main package</td>
<td>Separate download</td>
</tr>
<tr>
<td>After installation: <code>&lt;ipp cryptography directory&gt;/lib/&lt;arch&gt;</code></td>
<td>After installation: <code>&lt;ipp cryptography directory&gt;/lib/&lt;arch&gt;/threaded</code></td>
<td></td>
</tr>
<tr>
<td><strong>Static linking</strong></td>
<td>Windows® OS: <code>mt</code> suffix in a library name (<code>ippcpmt.lib</code>)</td>
<td>Windows® OS: <code>mt</code> suffix in a library name (<code>ippcpmt.lib</code>)</td>
</tr>
<tr>
<td>Linux® OS and macOS*: no suffix in a library name (<code>libippcp.a</code>)</td>
<td>Linux® OS and macOS*: no suffix in a library name (<code>libippcp.a</code>)</td>
<td></td>
</tr>
<tr>
<td><strong>Dynamic Linking</strong></td>
<td>Default (no suffix)</td>
<td>Default (no suffix)</td>
</tr>
</tbody>
</table>
To switch between single- or multi-threaded Intel IPP Cryptography libraries, set the path to the preferred library in system variables or in your project, for example:

- **Windows* OS:**
  - Single-threaded: SET LIB=<ipp cryptography directory>/lib/<arch>
  - Multi-threaded: SET LIB=<ipp cryptography directory>/lib/<arch>/threaded

- **Linux* OS/macOS*:
  - Single-threaded: gcc <options> -L <ipp cryptography directory>/lib/<arch>
  - Multi-threaded: gcc <options> -L <ipp cryptography directory>/lib/<arch>/threaded

**NOTE**
On Linux* OS and macOS* Intel IPP Cryptography library depends on the Intel® C++ Compiler 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
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 **IncludeDirectories**, 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 `C:\Program Files (x86)\IntelSWTools\compilers_and_libraries_<version>\windows\ippcp\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 `C:\Program Files (x86)\IntelSWTools\compilers_and_libraries_<version>\windows\ippcp\lib\<arch>_win`, 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 `;<install_dir>\redist\<arch>\ippcp\` 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 `ippcvars.bat <arch> from the directory <ipp cryptography directory>\bin\` 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
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.

Many factors may affect Intel IPP Cryptography performance. One of the best ways 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:

<table>
<thead>
<tr>
<th>Group</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set optimization layer to test</td>
<td><code>-T[cpu-features]</code></td>
<td>Call <code>ippcpSetCpuFeatures</code></td>
</tr>
<tr>
<td>Report Configuration</td>
<td>`-A&lt;Timing</td>
<td>Params</td>
</tr>
<tr>
<td></td>
<td><code>-o[&lt;file-name&gt;]</code></td>
<td>Create <code>&lt;file-name&gt;.txt</code> file and write console output to it</td>
</tr>
<tr>
<td></td>
<td><code>-O[&lt;file-name&gt;]</code></td>
<td>Add console output to the file <code>&lt;file-name&gt;.txt</code></td>
</tr>
<tr>
<td></td>
<td>`-L &lt;ERR</td>
<td>WARN</td>
</tr>
<tr>
<td></td>
<td><code>-r[&lt;file-name&gt;]</code></td>
<td>Create <code>&lt;file-name&gt;.csv</code> file and write perfsys results to it</td>
</tr>
<tr>
<td></td>
<td><code>-R[&lt;file-name&gt;]</code></td>
<td>Add test results to the file <code>&lt;file-name&gt;.csv</code></td>
</tr>
<tr>
<td></td>
<td><code>-q[&lt;file-name&gt;]</code></td>
<td>Create <code>&lt;file-name&gt;.csv</code> and write function parameter name lines to it</td>
</tr>
<tr>
<td></td>
<td><code>-q+</code></td>
<td>Add function parameter name lines to perfsys results table file</td>
</tr>
<tr>
<td>Group</td>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Set function parameters</td>
<td>-Q</td>
<td>Exit after creation of the function parameter name table</td>
</tr>
<tr>
<td></td>
<td>-u[&lt;file-name&gt;]</td>
<td>Create &lt;file-name&gt;.csv file and write summary table ('_sum' is added to default file name)</td>
</tr>
<tr>
<td></td>
<td>-U[&lt;file-name&gt;]</td>
<td>Add summary table to the file &lt;file-name&gt;.csv ('_sum' is added to default file name)</td>
</tr>
<tr>
<td></td>
<td>-g[&lt;file-name&gt;]</td>
<td>Create signal file at the end of the whole testing</td>
</tr>
<tr>
<td></td>
<td>-l&lt;dir-name&gt;</td>
<td>Set default directory for output files</td>
</tr>
<tr>
<td></td>
<td>-k&lt;and</td>
<td>or&gt;</td>
</tr>
<tr>
<td></td>
<td>-F&lt;func-name&gt;</td>
<td>Start testing from function with func-name full name</td>
</tr>
<tr>
<td></td>
<td>-Y&lt;HIGH/NORMAL&gt;</td>
<td>Set high or normal process priority (normal is default)</td>
</tr>
<tr>
<td></td>
<td>-H[ONLY]</td>
<td>Add 'Interest' column to .csv file [and run only hot tests]</td>
</tr>
<tr>
<td></td>
<td>-N&lt;num-threads&gt;</td>
<td>Call ippcpSetNumThreads(&lt;num-threads&gt;)</td>
</tr>
<tr>
<td></td>
<td>-s[-]</td>
<td>Sort or do not sort functions (sort mode is default)</td>
</tr>
<tr>
<td></td>
<td>-e</td>
<td>Enumerate tests and exit</td>
</tr>
<tr>
<td></td>
<td>-v</td>
<td>Display the version number of the perfsys and exit</td>
</tr>
<tr>
<td></td>
<td>-@&lt;file-name&gt;</td>
<td>Read command-line options for the specified file</td>
</tr>
<tr>
<td></td>
<td>-d&lt;name&gt;=&lt;value&gt;</td>
<td>Set perfsys parameter value</td>
</tr>
<tr>
<td></td>
<td>-i[&lt;file-name&gt;]</td>
<td>Read perfsys parameters from the file &lt;file-name&gt;.ini</td>
</tr>
<tr>
<td></td>
<td>-I[&lt;file-name&gt;]</td>
<td>Write perfsys parameters to the file &lt;file-name&gt;.ini and exit</td>
</tr>
<tr>
<td></td>
<td>-P</td>
<td>Read tested function names from the .ini file</td>
</tr>
<tr>
<td></td>
<td>-n&lt;title-name&gt;</td>
<td>Set default title name for .ini file and output files</td>
</tr>
<tr>
<td></td>
<td>-p&lt;dir-name&gt;</td>
<td>Set default directory for .ini file and input test data files</td>
</tr>
<tr>
<td>Select functions</td>
<td>-f &lt;or-pattern&gt;</td>
<td>Run tests for functions with pattern in their names, case sensitive</td>
</tr>
<tr>
<td></td>
<td>-f-&lt;not-pattern&gt;</td>
<td>Do not test functions with pattern in their names, case sensitive</td>
</tr>
<tr>
<td>Group</td>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>-f+&lt;and-pattern&gt;</td>
<td>Run tests only for functions with pattern in their names, case sensitive</td>
</tr>
<tr>
<td></td>
<td>-f=&lt;eq-pattern&gt;</td>
<td>Run tests for functions with pattern full name</td>
</tr>
<tr>
<td></td>
<td>-t[-</td>
<td>+]&lt;pattern&gt;</td>
</tr>
<tr>
<td></td>
<td>-m[-</td>
<td>+]&lt;pattern&gt;</td>
</tr>
<tr>
<td>Help</td>
<td>-h</td>
<td>Display short help and exit</td>
</tr>
<tr>
<td></td>
<td>-hh</td>
<td>Display extended help and exit</td>
</tr>
<tr>
<td></td>
<td>-h&lt;key&gt;</td>
<td>Display extended help for the key and exit</td>
</tr>
</tbody>
</table>
Appendix B: Intel® IPP
Cryptography Threading and OpenMP* Support

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.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for
optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and
SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or
effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-
dependent optimizations in this product are intended for use with Intel microprocessors. Certain
optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to
the applicable product User and Reference Guides for more information regarding the specific instruction
sets covered by this notice.

Notice revision #20110804

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.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for
optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and
SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or
effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-
dependent optimizations in this product are intended for use with Intel microprocessors. Certain
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.