OpenSSL 3.0

OpenSSL 3.0 is the next release of OpenSSL that is currently in development. This page is intended as a collection of notes for people downloading the alpha/beta releases or who are planning to upgrade from a previous version of OpenSSL to 3.0.

**Contents**

 [[hide](https://wiki.openssl.org/index.php/OpenSSL_3.0)]

* [1Main Changes in OpenSSL 3.0 from OpenSSL 1.1.1](https://wiki.openssl.org/index.php/OpenSSL_3.0#Main_Changes_in_OpenSSL_3.0_from_OpenSSL_1.1.1)
  + [1.1Major Release](https://wiki.openssl.org/index.php/OpenSSL_3.0#Major_Release)
  + [1.2License Change](https://wiki.openssl.org/index.php/OpenSSL_3.0#License_Change)
  + [1.3Providers and FIPS support](https://wiki.openssl.org/index.php/OpenSSL_3.0#Providers_and_FIPS_support)
  + [1.4Low Level APIs](https://wiki.openssl.org/index.php/OpenSSL_3.0#Low_Level_APIs)
  + [1.5Legacy Algorithms](https://wiki.openssl.org/index.php/OpenSSL_3.0#Legacy_Algorithms)
  + [1.6Engines and "METHOD" APIs](https://wiki.openssl.org/index.php/OpenSSL_3.0#Engines_and_.22METHOD.22_APIs)
  + [1.7Versioning Scheme](https://wiki.openssl.org/index.php/OpenSSL_3.0#Versioning_Scheme)
  + [1.8Other major new features](https://wiki.openssl.org/index.php/OpenSSL_3.0#Other_major_new_features)
  + [1.9Other notable deprecations and changes](https://wiki.openssl.org/index.php/OpenSSL_3.0#Other_notable_deprecations_and_changes)
* [2Installation and Compilation of OpenSSL 3.0](https://wiki.openssl.org/index.php/OpenSSL_3.0#Installation_and_Compilation_of_OpenSSL_3.0)
* [3Upgrading to OpenSSL 3.0 from OpenSSL 1.1.1](https://wiki.openssl.org/index.php/OpenSSL_3.0#Upgrading_to_OpenSSL_3.0_from_OpenSSL_1.1.1)
* [4Upgrading to OpenSSL 3.0 from OpenSSL 1.0.2](https://wiki.openssl.org/index.php/OpenSSL_3.0#Upgrading_to_OpenSSL_3.0_from_OpenSSL_1.0.2)
  + [4.1Upgrading from the OpenSSL 2.0 FIPS Object Module](https://wiki.openssl.org/index.php/OpenSSL_3.0#Upgrading_from_the_OpenSSL_2.0_FIPS_Object_Module)
* [5Completing the installation of the FIPS Module](https://wiki.openssl.org/index.php/OpenSSL_3.0#Completing_the_installation_of_the_FIPS_Module)
* [6Programming in OpenSSL 3.0](https://wiki.openssl.org/index.php/OpenSSL_3.0#Programming_in_OpenSSL_3.0)
  + [6.1Library Contexts](https://wiki.openssl.org/index.php/OpenSSL_3.0#Library_Contexts)
  + [6.2Providers](https://wiki.openssl.org/index.php/OpenSSL_3.0#Providers)
  + [6.3Fetching algorithms and property queries](https://wiki.openssl.org/index.php/OpenSSL_3.0#Fetching_algorithms_and_property_queries)
* [7Using the FIPS Module in applications](https://wiki.openssl.org/index.php/OpenSSL_3.0#Using_the_FIPS_Module_in_applications)
  + [7.1Making all applications use the FIPS module by default](https://wiki.openssl.org/index.php/OpenSSL_3.0#Making_all_applications_use_the_FIPS_module_by_default)
  + [7.2Selectively making applications use the FIPS module by default](https://wiki.openssl.org/index.php/OpenSSL_3.0#Selectively_making_applications_use_the_FIPS_module_by_default)
  + [7.3Programmatically loading the FIPS module (default library context)](https://wiki.openssl.org/index.php/OpenSSL_3.0#Programmatically_loading_the_FIPS_module_.28default_library_context.29)
  + [7.4Loading the FIPS module at the same time as other providers](https://wiki.openssl.org/index.php/OpenSSL_3.0#Loading_the_FIPS_module_at_the_same_time_as_other_providers)
  + [7.5Programmatically loading the FIPS module (non-default library context)](https://wiki.openssl.org/index.php/OpenSSL_3.0#Programmatically_loading_the_FIPS_module_.28non-default_library_context.29)
  + [7.6Using Serializers with the FIPS module](https://wiki.openssl.org/index.php/OpenSSL_3.0#Using_Serializers_with_the_FIPS_module)
  + [7.7Using the FIPS module in SSL/TLS](https://wiki.openssl.org/index.php/OpenSSL_3.0#Using_the_FIPS_module_in_SSL.2FTLS)
  + [7.8Confirming that an algorithm is being provided by the FIPS module](https://wiki.openssl.org/index.php/OpenSSL_3.0#Confirming_that_an_algorithm_is_being_provided_by_the_FIPS_module)
* [8Openssl command line application changes](https://wiki.openssl.org/index.php/OpenSSL_3.0#Openssl_command_line_application_changes)
* [9STATUS of current development](https://wiki.openssl.org/index.php/OpenSSL_3.0#STATUS_of_current_development)
  + [9.1Known issues](https://wiki.openssl.org/index.php/OpenSSL_3.0#Known_issues)
    - [9.1.1Building and testing](https://wiki.openssl.org/index.php/OpenSSL_3.0#Building_and_testing)
    - [9.1.2Integration](https://wiki.openssl.org/index.php/OpenSSL_3.0#Integration)
    - [9.1.3Programming](https://wiki.openssl.org/index.php/OpenSSL_3.0#Programming)
    - [9.1.4SSL/TLS](https://wiki.openssl.org/index.php/OpenSSL_3.0#SSL.2FTLS)
  + [9.2Platforms](https://wiki.openssl.org/index.php/OpenSSL_3.0#Platforms)
  + [9.3Features](https://wiki.openssl.org/index.php/OpenSSL_3.0#Features)
    - [9.3.1Provider implemented operation types](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_operation_types)
    - [9.3.2Provider implemented ciphers](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_ciphers)
    - [9.3.3Provider implemented digests](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_digests)
    - [9.3.4Provider implemented MACs](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_MACs)
    - [9.3.5Provider implemented KDFs](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_KDFs)
    - [9.3.6Provider implemented asymmetric key types](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_asymmetric_key_types)
    - [9.3.7Provider implemented asymmetric ciphers](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_asymmetric_ciphers)
    - [9.3.8Provider implemented signature](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_signature)
    - [9.3.9Provider implemented key exchange](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_key_exchange)
    - [9.3.10Provider implemented serializers / deserializers](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_serializers_.2F_deserializers)
      * [9.3.10.1Serializers](https://wiki.openssl.org/index.php/OpenSSL_3.0#Serializers)
      * [9.3.10.2Deserializers](https://wiki.openssl.org/index.php/OpenSSL_3.0#Deserializers)
    - [9.3.11Provider implemented OSSL\_STORE URI schemes](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implemented_OSSL_STORE_URI_schemes)
  + [9.4Library Context/Provider implementation support in other OpenSSL APIs](https://wiki.openssl.org/index.php/OpenSSL_3.0#Library_Context.2FProvider_implementation_support_in_other_OpenSSL_APIs)

1 Main Changes in OpenSSL 3.0 from OpenSSL 1.1.1[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=1)]

**1.1 Major Release**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=2)]

OpenSSL 3.0 is a major release and consequently any application that currently uses an older version of OpenSSL will at the very least need to be recompiled in order to work with the new version. It is the intention that the large majority of applications will work unchanged with OpenSSL 3.0 if those applications previously worked with OpenSSL 1.1.1. However this is not guaranteed and some changes may be required in some cases. Changes may also be required if applications need to take advantage of some of the new features available in OpenSSL 3.0 such as the availability of the FIPS module.

**1.2 License Change**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=3)]

In previous versions, OpenSSL was licensed under the dual [OpenSSL and SSLeay licenses](https://www.openssl.org/source/license-openssl-ssleay.txt) (both licenses apply). From OpenSSL 3.0 this is replaced by the [Apache License v2](https://www.openssl.org/source/apache-license-2.0.txt).

**1.3 Providers and FIPS support**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=4)]

One of the key changes from OpenSSL 1.1.1 is the introduction of the Provider concept. Providers collect together and make available algorithm implementations. With OpenSSL 3.0 it is possible to specify, either programmatically or via a config file, which providers you want to use for any given application. OpenSSL 3.0 comes with 4 different providers as standard. Over time third parties may distribute additional providers that can be plugged into OpenSSL. All algorithm implementations available via providers are accessed through the "EVP" set of APIs. They cannot be accessed using the "low level" APIs (see below).

**1.4 Low Level APIs**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=5)]

OpenSSL has historically provided two sets of APIs for invoking cryptographic algorithms: the "EVP" APIs and the "low level" APIs. The EVP APIs are typically designed to work across all algorithm types. The "low level" APIs are targeted at a specific algorithm implementation. For example, the EVP APIs provide the functions `EVP\_EncryptInit\_ex`, `EVP\_EncryptUpdate` and `EVP\_EncryptFinal` to perform symmetric encryption. Those functions can be used with the algorithms AES, CHACHA, 3DES etc. On the other hand to do AES encryption using the low level APIs you would have to call AES specific functions such as `AES\_set\_encrypt\_key`, `AES\_encrypt`, and so on. The functions for 3DES are different.

Use of the low level APIs has been informally discouraged by the OpenSSL development team for a long time. However in OpenSSL 3.0 this is made more formal. All such low level APIs have been deprecated. You may still *use* them in your applications, but you may start to see deprecation warnings during compilation (dependent on compiler support for this). Deprecated APIs may be removed from future versions of OpenSSL so you are strongly encouraged to update your code to use the EVP APIs instead.

**1.5 Legacy Algorithms**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=6)]

Some cryptographic algorithms that were available via the EVP APIs are now considered legacy and their use is strongly discouraged. These legacy EVP algorithms are still available in OpenSSL 3.0 but not by default. If you want to use them then you must load the legacy provider. This can be as simple as a config file change, or can be done programmatically (see below).

**1.6 Engines and "METHOD" APIs**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=7)]

The refactoring to support Providers conflicts internally with the APIs used to support engines, including the ENGINE API and any function that creates or modifies custom "METHODS" (for example EVP\_MD\_meth\_new, EVP\_CIPHER\_meth\_new, EVP\_PKEY\_meth\_new, RSA\_meth\_new, EC\_KEY\_METHOD\_new, etc.). These functions are being deprecated in OpenSSL 3.0, and users of these APIs should know that their use can likely bypass provider selection and configuration, with unintended consequences. This is particularly relevant for applications written to use the OpenSSL 3.0 FIPS module, as detailed below. Authors and maintainers of external engines are strongly encouraged to refactor their code transforming engines into providers using the new Provider API and avoiding deprecated methods.

**1.7 Versioning Scheme**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=8)]

The OpenSSL versioning scheme has changed with the 3.0 release. The new versioning scheme has this format:

MAJOR.MINOR.PATCH

For version 1.1.1 and below different patch levels were indicated by a letter at the end of the release version number. This will no longer be used and instead the patch level is indicated by the final number in the version. A change in the second (MINOR) number indicates that new features may have been added. OpenSSL versions with the same major number are API and ABI compatible. If the major number changes then API and ABI compatibility is not guaranteed.

**1.8 Other major new features**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=9)]

* Implementation of the Certificate Management Protocol (CMP, [RFC 4210](https://tools.ietf.org/html/rfc4210)) also covering CRMF ([RFC 4211](https://tools.ietf.org/html/rfc4211)) and HTTP transfer ([RFC 6712](https://tools.ietf.org/html/rfc6712))
* A proper HTTP(S) client in libcrypto supporting GET and POST, redirection, plain and ASN.1-encoded contents, proxies, and timeouts
* EVP\_KDF APIs have been introduced for working with Key Derivation Functions
* EVP\_MAC APIs have been introduced for working with MACs
* Support for Linux Kernel TLS

**1.9 Other notable deprecations and changes**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=10)]

* The function code part of an OpenSSL error code is no longer relevant and is always set to zero. Related functions are deprecated.
* The STACK and HASH macro's have been cleaned up, so that the type-safe wrappers are declared everywhere and implemented once. See the manpage at <https://www.openssl.org/docs/manmaster/man3/DEFINE_STACK_OF.html> for stack, and hopefully soon once the PR is merged, <https://www.openssl.org/docs/manmaster/man3/DECLARE_LHASH_OF.html> (but not yet as of this writing).

2 Installation and Compilation of OpenSSL 3.0[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=11)]

Please refer to the INSTALL.md file in the top of the distribution for instructions on how to build and install OpenSSL 3.0. Please also refer to the various platform specific NOTES files for your specific platform.

NOTE: The OpenSSL 3.0 alpha 1 release contains an error introduced during the release process which results in a failed compilation. There are two workarounds to choose between:

* apply [the patch from github PR #11624](https://github.com/openssl/openssl/pull/11624/files).
* edit the VERSION file in the top of the distribution to remove the quotes around the date on the RELEASE\_DATE line, i.e. make that line look like this:

RELEASE\_DATE=23 Apr 2020

3 Upgrading to OpenSSL 3.0 from OpenSSL 1.1.1[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=12)]

Upgrading to OpenSSL 3.0 from OpenSSL 1.1.1 should be relatively straight forward in most cases. The most likely area where you will encounter problems is if you have used low level APIs in your code (as discussed above). In that case you are likely to start seeing deprecation warnings when compiling your application. If this happens you have 3 options:

1) Ignore the warnings. They are just warnings. The deprecated functions are still present and you may still use them. However be aware that they may be removed from a future version of OpenSSL.

2) Suppress the warnings. Refer to your compiler documentation on how to do this.

3) Remove your usage of the low level APIs. In this case you will need to rewrite your code to use the EVP APIs instead.

4 Upgrading to OpenSSL 3.0 from OpenSSL 1.0.2[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=13)]

Upgrading to OpenSSL 3.0 from OpenSSL 1.0.2 is likely to be significantly more difficult. In addition to the issues discussed above in the section about upgrading from 1.1.1, the main things to be aware of are:

1) The build and installation procedure has changed significantly since OpenSSL 1.0.2. Check the file INSTALL.md in the top of the installation for instructions on how to build and install OpenSSL for your platform. Also checkout the various NOTES files in the same directory, as applicable for your platform.

2) Many structures have been made opaque in OpenSSL 3.0. The structure definitions have been removed from the public header files and moved to internal header files. In practice this means that you can no longer stack allocate some structures. Instead they must be heap allocated through some function call (typically those function names have a `\_new` suffix to them). Additionally you must use "setter" or "getter" functions to access the fields within those structures.

For example code that previously looked like this:

EVP\_MD\_CTX md\_ctx;

EVP\_MD\_CTX\_init(&md\_ctx);

/\* Do something with the md\_ctx \*/

will now generate compiler errors. For example:

md\_ctx.c:6:16: error: storage size of ‘md\_ctx’ isn’t known

The code needs to be amended to look like this:

EVP\_MD\_CTX \*md\_ctx;

md\_ctx = EVP\_MD\_CTX\_new();

if (md\_ctx == NULL)

/\* Error \*/;

/\* Do something with the md\_ctx \*/

EVP\_MD\_CTX\_free(md\_ctx);

3) Support for TLSv1.3 has been added which has a number of implications for SSL/TLS applications. See the [TLS1.3](https://wiki.openssl.org/index.php/TLS1.3) page for further details.

More details about the breaking changes between OpenSSL versions 1.0.2 and 1.1.0 can be found on the [OpenSSL 1.1.0 Changes](https://wiki.openssl.org/index.php/OpenSSL_1.1.0_Changes) page.

**4.1 Upgrading from the OpenSSL 2.0 FIPS Object Module**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=14)]

The OpenSSL 2.0 FIPS Object Module was a separate download that had to be built separately and then integrated into your main OpenSSL 1.0.2 build. In OpenSSL 3.0 the FIPS support is fully integrated into the mainline version of OpenSSL and is no longer a separate download. You do not need to take separate build steps to add the FIPS support - it is built by default. You *do* need to take steps to ensure that your application is *using* the FIPS module in OpenSSL 3.0. See the further notes below on configuring this.

The function calls 'FIPS\_mode()' and 'FIPS\_mode\_set()' have been removed from OpenSSL 3.0. You should rewrite your application to not use them. See the sections below on how to write applications to use the FIPS Module in OpenSSL 3.0.

5 Completing the installation of the FIPS Module[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=15)]

Once OpenSSL has been built and installed you will need to take explicit steps to complete the installation of the FIPS module (if you wish to use it). The OpenSSL 3.0 FIPS support is in the form of the FIPS provider which, on Unix, is in a `fips.so` file. On Windows this will be called `fips.dll`. Following installation of OpenSSL 3.0 the default location for this file is '/usr/local/lib/ossl-modules/fips.so' on Unix or 'C:\Program Files\OpenSSL\lib\ossl-modules\fips.dll' on Windows.

To complete the installation you need to run the 'fipsinstall' command line application. This does 2 things:

* Runs the FIPS module self tests
* Generates FIPS module config file output containing information about the module such as the self test status, and the module checksum

The FIPS module *must* have the self tests run, and the FIPS module config file output generated on *every* machine that it is to be used on. You **must not** copy the FIPS module config file output data from one machine to another.

For example, to install the FIPS module to its default location:

$ openssl fipsinstall -out /usr/local/ssl/fipsmodule.cnf -module /usr/local/lib/ossl-modules/fips.so -provider\_name fips -mac\_name HMAC -macopt digest:SHA256 -macopt hexkey:00 -section\_name fips\_sect

If you installed OpenSSL to a different location, you need to adjust the output and module path accordingly.

6 Programming in OpenSSL 3.0[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=16)]

Applications written to work with OpenSSL 1.1.1 will mostly just work with OpenSSL 3.0. However changes will be required if you want to take advantage of some of the new features that OpenSSL 3.0 makes available. In order to do that you need to understand some new concepts introduced in OpenSSL 3.0.

**6.1 Library Contexts**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=17)]

A library context can be thought of as a "scope" for OpenSSL operations. All functionality operates with the scope of a library context. Multiple library contexts may exist at the same time, and they each may be configured differently. A library context is represented by the newly introduced OPENSSL\_CTX type. See the man page [here](https://www.openssl.org/docs/manmaster/man3/OPENSSL_CTX.html).

Many new functions have been introduced into OpenSSL that take an OPENSSL\_CTX parameter. In many cases these are variants of some other function that existed in 1.1.1 and work in much the same way - except that they now operate within the scope of the given library context.

All applications have available to them the "default library context". This library context always exists and, if you don't otherwise specify one, this is the library context that will be used. Any function that takes an OPENSSL\_CTX value as a parameter will accept the value NULL for that parameter in order to refer to the default library context. You can also explicitly create new ones via the OPENSSL\_CTX\_new() function. See the man page for further details.

Config files affect a given library context. It is quite possible to have multiple library contexts in use, with each one having been configured with a different config file (see the OPENSSL\_CTX\_load\_config() function described on the man page).

**6.2 Providers**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=18)]

Providers are containers for algorithm implementations. Whenever a cryptographic algorithm is used via the EVP APIs a provider is selected. It is that provider implementation that actually does the required work. There are four providers distributed with OpenSSL. In the future we expect third parties to distribute their own providers which can be added to OpenSSL dynamically. Documentation about writing providers is available on the man page [here](https://www.openssl.org/docs/manmaster/man7/provider.html).

The standard providers are:

* The default provider. This collects together all of the standard built-in OpenSSL algorithm implementations. If an application doesn't specify anything else explicitly (e.g. in the application or via config), then this is the provider that will be used. It is loaded automatically the first time that we try to get an algorithm from a provider if no other provider has been loaded yet. If another provider has already been loaded then it won't be loaded automatically. Therefore if you want to use it in conjunction with other providers then you must load it explicitly. This is a "built-in" provider which means that it is built into libcrypto and does not exist as a separate standalone module.
* The legacy provider. This is a collection of legacy algorithms that are either no longer in common use or strongly discouraged from use. However some applications may need to use these algorithms for backwards compatibility reasons. This provider is NOT loaded by default. This may mean that some applications upgrading from earlier versions of OpenSSL may find that some algorithms are no longer available unless they load the legacy provider explicitly. Algorithms in the legacy provider include MD2, MD4, MDC2, RMD160, CAST5, BF (Blowfish), IDEA, SEED, RC2, RC4, RC5 and DES (but not 3DES).
* The FIPS provider. This contains a sub-set of the algorithm implementations available from the default provider. Algorithms available in this provider conform to FIPS standards. It is intended that this provider will be FIPS140-2 validated. In some cases there may be minor behavioural differences between algorithm implementations in this provider compared to the equivalent algorithm in the default provider. This is typically in order to conform to FIPS standards.
* The null provider. This provider is "built-in" to libcrypto and contains no algorithm implementations. In order to guarantee that the default provider is not automatically loaded, the null provider can be loaded instead. This can be useful if you are using non-default library contexts and want to ensure that the default library context is never used "by accident".

Providers to be loaded can be specified in the OpenSSL config file. See the man page [here](https://www.openssl.org/docs/manmaster/man5/config.html)for information about how to configure providers via the config file, and how to automatically activate them. This is a minimal config file example to load and activate both the legacy and the default provider in the default library context.

openssl\_conf = openssl\_init

[openssl\_init]

providers = provider\_sect

[provider\_sect]

default = default\_sect

legacy = legacy\_sect

[default\_sect]

activate = 1

[legacy\_sect]

activate = 1

It is also possible to load them programmatically. For example you can load the legacy provider into the default library context as shown below. Note that once you have explicitly loaded a provider into the library context the default provider will no longer be automatically loaded. Therefore you will often also want to explicitly load the default provider, as is done here:

#include <stdio.h>

#include <stdlib.h>

#include <openssl/provider.h>

int main(void)

{

OSSL\_PROVIDER \*legacy;

OSSL\_PROVIDER \*deflt;

/\* Load Multiple providers into the default (NULL) library context \*/

legacy = OSSL\_PROVIDER\_load(NULL, "legacy");

if (legacy == NULL) {

printf("Failed to load Legacy provider\n");

exit(EXIT\_FAILURE);

}

deflt = OSSL\_PROVIDER\_load(NULL, "default");

if (deflt == NULL) {

printf("Failed to load Default provider\n");

OSSL\_PROVIDER\_unload(legacy);

exit(EXIT\_FAILURE);

}

/\* Rest of application \*/

OSSL\_PROVIDER\_unload(legacy);

OSSL\_PROVIDER\_unload(deflt);

exit(EXIT\_SUCCESS);

}

**6.3 Fetching algorithms and property queries**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=19)]

In order to use a cryptographic algorithm (such as AES) then an implementation for it must first be "fetched" from the available providers that have been loaded into the library context being used. This can be done either implicitly or explicitly.

With implicit fetching the application does not need to do anything special. Algorithms implementations will be fetched automatically by the relevant APIs. For example:

EVP\_MD\_CTX \*mdctx;

mdctx = EVP\_MD\_CTX\_new();

if (mdctx == NULL)

goto err;

if (EVP\_DigestInit\_ex(mdctx, EVP\_sha256(), NULL) != 1)

goto err;

In this code we are initialising a digest operation to use the SHA256 algorithm. The EVP\_DigestInit\_ex() function will automatically fetch an implementation of the SHA256 algorithm from the available providers when it needs to. It will do so using the default library context and the default property query string (see below).

With explicit fetching an application fetches the implementation to be used up front, and then passes that to the relevant EVP API. For example:

EVP\_MD\_CTX \*mdctx;

EVP\_MD \*sha256;

mdctx = EVP\_MD\_CTX\_new();

if (mdctx == NULL)

goto err;

/\*

\* Setting the library ctx to NULL here fetches the algorithm from the providers loaded

\* into the default library context

\*/

sha256 = EVP\_MD\_fetch(NULL, "SHA2-256", NULL);

if (sha256 == NULL)

goto err;

if (EVP\_DigestInit\_ex(mdctx, sha256, NULL) != 1)

goto err;

/\* Explicit fetches return a dynamic object that must be freed \*/

EVP\_MD\_free(sha256);

In this example we have explicitly fetched an implementation of SHA256 from the set of available providers loaded into the default library context.

With an explicit fetch we can additionally supply a property query to further specify which implementation we wish to obtain. For example:

sha256 = EVP\_MD\_fetch(NULL, "SHA2-256", "fips=yes");

Here we are explicitly fetching a FIPS validated implementation of the SHA256 algorithm. Such an implementation exists in the FIPS provider, so we would need to have ensured that the FIPS provider was loaded into the default library context in order for this to be successful. If no algorithm implementation that matches the criteria can be located then the fetch will fail.

See the section on fetching algorithms in the provider man page for further details: [[1]](https://www.openssl.org/docs/manmaster/man7/provider.html#Fetching-algorithms).

If no specific property query is required then NULL can be passed for the last argument. In any case any supplied property query is combined with the default property query. If nothing else is specified then the default property query is empty. However this can be changed so that every fetch automatically inherits these default properties. Default properties can either be set programmatically or via a config file. See the section [Loading the FIPS module at the same time as other providers](https://wiki.openssl.org/index.php/OpenSSL_3.0#Loading_the_FIPS_module_at_the_same_time_as_other_providers) for an example of how to do this.

Note that default properties are not currently functional in the OpenSSL 3.0 alpha 1 or alpha 2 releases.

7 Using the FIPS Module in applications[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=20)]

There are a number of different ways that OpenSSL can be used in conjunction with the FIPS module. Which is the correct approach to use will depend on your own specific circumstances and what you are attempting to achieve. Note that the old functions FIPS\_mode() and FIPS\_mode\_set() are no longer present so you must remove them from your application if you use them.

**7.1 Making all applications use the FIPS module by default**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=21)]

One simple approach is to cause all applications that are using OpenSSL to only use the FIPS module for cryptographic algorithms by default.

This approach can be done purely via configuration. As long as applications are built and linked against OpenSSL 3.0 and do not override the loading of the default config file or its settings then they will automatically start using the FIPS module without the need for any further code changes.

To do this the default OpenSSL config file will have to be modified. The location of this config file will depend on the platform, and any options that were given during the build process. You can check the location of the config file by running this command:

$ openssl version -d

OPENSSLDIR: "/usr/local/ssl"

Caution: Many Operating Systems install OpenSSL by default. It is a common error to not have the correct version of OpenSSL on your $PATH. Check that you are running an OpenSSL 3.0 version like this:

$ openssl version -v

OpenSSL 3.0.0-dev xx XXX xxxx (Library: OpenSSL 3.0.0-dev xx XXX xxxx)

The OPENSSLDIR value above gives the directory name for where the default config file is stored. So in this case the default config file will be called /usr/local/ssl/openssl.cnf

Edit the config file to add the following lines near the beginning:

openssl\_conf = openssl\_init

.include /usr/local/ssl/fipsmodule.cnf

[openssl\_init]

providers = provider\_sect

[provider\_sect]

fips = fips\_sect

Obviously the include file location above should match the name of the FIPS module config file that you installed earlier.

Any applications that use OpenSSL 3.0 and are started after these changes are made will start using only the FIPS module unless those applications take explicit steps to avoid this default behaviour.

This approach has the primary advantage that it is simple, and no code changes are required in applications in order to benefit from the FIPS module. There are some disadvantages to this approach:

* You may not want *all* applications to use the FIPS module. It may be the case that some applications should and some should not.
* If applications take explicit steps to not load the default config file or set different settings then this method will not work for them
* The algorithms available in the FIPS module are a subset of the algorithms that are available in the default OpenSSL Provider. If those applications attempt to use any algorithms that are not present, then they will fail.
* Usage of certain APIs avoids the use of the FIPS module. If any applications use those APIs then the FIPS module will not be used.

**7.2 Selectively making applications use the FIPS module by default**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=22)]

A variation on the above approach is to do the same thing on an individual application basis. The default OpenSSL config file depends on the compiled in value for OPENSSLDIR as described in the section above. However it is also possible to override the config file to be used via the OPENSSL\_CONF environment variable. For example the following on Unix will cause the application to be executed with a non-standard config file location:

$ OPENSSL\_CONF=/my/non-default/openssl.cnf myapplication

Using this mechanism you can control which config file is loaded (and hence whether the FIPS module is loaded) on an application by application basis.

This removes the disadvantage listed above that you may not want all applications to use the FIPS module. All the other advantages and disadvantages still apply.

**7.3 Programmatically loading the FIPS module (default library context)**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=23)]

Applications may choose to load the FIPS provider explicitly rather than relying on config to do this. The config file is still necessary in order to hold the FIPS module config data (such as its self test status and integrity data). But in this case we do not automatically activate the FIPS provider via that config file.

To do things this way configure as per the section "Making all applications use the FIPS module by default" above, but edit the fipsmodule.cnf file to remove or comment out the line which says "activate = 1". This means all the required config information will be available to load the FIPS module, but it is not actually automatically loaded when the application starts. The FIPS provider can then be loaded programmatically like this:

#include <openssl/provider.h>

int main(void)

{

OSSL\_PROVIDER \*fips;

fips = OSSL\_PROVIDER\_load(NULL, "fips");

if (fips == NULL) {

printf("Failed to load FIPS provider\n");

exit(EXIT\_FAILURE);

}

/\* Rest of application \*/

OSSL\_PROVIDER\_unload(fips);

exit(EXIT\_SUCCESS);

}

Note that this should be one of the first things that you do in your application. If any OpenSSL functions get called that require the use of cryptographic functions before this occurs then, if no provider has yet been loaded, then the default provider will be automatically loaded. If you then later explicitly load the FIPS provider then you will have both the FIPS and the default provider loaded at the same time. It is undefined which implementation of an algorithm will be used if multiple implementations are available and you have not explicitly specified via a property query (see below) which one should be used.

Applications written to use the OpenSSL 3.0 FIPS module should not use any legacy APIs or features that avoid the FIPS module. Specifically this includes:

* Low level cryptographic APIs (use the EVP APIs instead). All such APIs are deprecated in OpenSSL 3.0 - so a simple rule is to avoid using all deprecated functions.
* Engines
* Any functions that create or modify custom "METHODS" (for example EVP\_MD\_meth\_new, EVP\_CIPHER\_meth\_new, EVP\_PKEY\_meth\_new, RSA\_meth\_new, EC\_KEY\_METHOD\_new, etc.)

**7.4 Loading the FIPS module at the same time as other providers**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=24)]

It is possible to have the FIPS provider and other providers (such as the default provider) all loaded at the same time into the same library context. You can use a property query string during algorithm fetches to specify which implementation you would like to use.

For example to fetch an implementation of SHA256 which conforms to FIPS standards you can specify the property query "fips=yes" like this:

EVP\_MD \*sha256;

sha256 = EVP\_MD\_fetch(NULL, "SHA2-256", "fips=yes");

If no property query is specified, or more than one implementation matches the property query then it is undefined which implementation of a particular algorithm will be returned.

This example shows an explicit request for an implementation of SHA256 from the default provider:

EVP\_MD \*sha256;

sha256 = EVP\_MD\_fetch(NULL, "SHA2-256", "provider=default");

It is also possible to set a default property query string. The following example sets the default property query of "fips=yes" for all fetches within the default library context:

EVP\_set\_default\_properties(NULL, "fips=yes");

NOTE: Default properties are currently not functional in the OpenSSL 3.0 alpha 1 and alpha 2 releases - see the known issues below

If a fetch function has both an explicit property query specified, and a default property query is defined then the two queries are merged together and both apply. It is also possible for a locally specified property query to override the default properties.

There are two important built-in properties that you should be aware of:

The "provider" property enables you to specify which provider you want an implementation to be fetched from, e.g. "provider=default" or "provider=fips". All algorithms implemented in a provider have this property set on them.

There is also the "fips" property. All FIPS algorithms match against the property query "fips=yes". There are also some non-cryptographic algorithms available in the default provider that also have the "fips=yes" property defined for them. These are the serializer algorithms that can (for example) be used to write out a key generated in the FIPS provider to a file. The serializer algorithms are not in the FIPS module itself but are allowed to be used in conjunction with the FIPS algorithms.

It is possible to specify default properties within a config file. For example the following config file automatically loads the default and fips providers and sets the default property value to be "fips=yes":

openssl\_conf = openssl\_init

.include /usr/local/ssl/fipsmodule.cnf

[openssl\_init]

providers = provider\_sect

alg\_section = algorithm\_sect

[provider\_sect]

fips = fips\_sect

default = default\_sect

[default\_sect]

activate = 1

[algorithm\_sect]

default\_properties = fips=yes

**7.5 Programmatically loading the FIPS module (non-default library context)**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=25)]

In addition to using properties to separate usage of the FIPS module from other usages this can also be achieved using library contexts. In this example we create two library contexts. In one we assume the existence of a config file called "openssl-fips.cnf" that automatically loads and configures the FIPS provider. The other library context will just use the default provider.

OPENSSL\_CTX \*fipslibctx, \*nonfipslibctx;

OSSL\_PROVIDER \*defctxnull = NULL;

EVP\_MD \*fipssha256 = NULL, \*nonfipssha256 = NULL;

int ret = 1;

/\*

\* Create two non-default library contexts. One for fips usage and one for

\* non-fips usage

\*/

fipslibctx = OPENSSL\_CTX\_new();

nonfipslibctx = OPENSSL\_CTX\_new();

if (fipslibctx == NULL || nonfipslibctx == NULL)

goto err;

/\* Prevent anything from using the default library context \*/

defctxnull = OSSL\_PROVIDER\_load(NULL, "null");

/\*

\* Load config file for the FIPS library context. We assume that this

\* config file will automatically activate the FIPS provider so we don't

\* need to explicitly load it here.

\*/

if (!OPENSSL\_CTX\_load\_config(fipslibctx, "openssl-fips.cnf"))

goto err;

/\*

\* We don't need to do anything special to load the default provider into

\* nonfipslibctx. This happens automatically if no other providers are

\* loaded. Because we don't call OPENSSL\_CTX\_load\_config() explicitly for

\* nonfipslibctx it will just use the default config file.

\*/

/\* As an example get some digests \*/

/\* Get a FIPS validated digest \*/

fipssha256 = EVP\_MD\_fetch(fipslibctx, "SHA2-256", NULL);

if (fipssha256 == NULL)

goto err;

/\* Get a non-FIPS validated digest \*/

nonfipssha256 = EVP\_MD\_fetch(nonfipslibctx, "SHA2-256", NULL);

if (nonfipssha256 == NULL)

goto err;

/\* Use the digests \*/

printf("Success\n");

ret = 0;

err:

EVP\_MD\_free(fipssha256);

EVP\_MD\_free(nonfipssha256);

OPENSSL\_CTX\_free(fipslibctx);

OPENSSL\_CTX\_free(nonfipslibctx);

OSSL\_PROVIDER\_unload(defctxnull);

return ret;

Note that we have made use of the special "null" provider here which we load into the default library context. We could have chosen to use the default library context for FIPS usage, and just create one additional library context for other usages - or vice versa. However if code has not been converted to use library contexts then the default library context will be automatically used. This could be the case for your own existing applications as well as certain parts of OpenSSL itself. Not all parts of OpenSSL are library context aware. If this happens then you could "accidentally" use the wrong library context for a particular operation. To be sure this doesn't happen you can load the "null" provider into the default library context. Because a provider has been explicitly loaded, the default provider will not automatically load. This means code using the default context by accident will fail because no algorithms will be available.

**7.6 Using Serializers with the FIPS module**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=26)]

Serializers are used to read and write keys or parameters from or to some external format (for example a PEM file). In the OpenSSL 3.0 alpha 1 and alpha 2 releases only the "write" serializers have been implemented. Reading will come in a later alpha release. If your application generates keys or parameters that then need to be written into PEM or DER format then it is likely that you will need to use a serializer to do this. In most cases this will be invisible to you if you are using APIs that existed in OpenSSL 1.1.1 or earlier such as i2d\_PrivateKey. However the appropriate serializer will need to be available in the library context associated with the key or parameter object. The built-in OpenSSL serializers are implemented in the default provider and are not in the FIPS module boundary. However since they are not cryptographic algorithms themselves it is still possible to use them in conjunction with the FIPS module, and therefore these serializers have the "fips=yes" property against them. You must ensure that the default provider is loaded into the library context in this case.

**7.7 Using the FIPS module in SSL/TLS**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=27)]

Writing an application that uses libssl in conjunction with the FIPS module is much the same as writing a normal libssl application. If you are using global properties to specify usage of FIPS validated algorithms then this will happen automatically for all cryptographic algorithms in libssl. If you are using a non-default library context to load the FIPS provider then you can supply this to libssl using the function SSL\_CTX\_new\_with\_libctx(). This works as a drop in replacement for the function SSL\_CTX\_new() except it provides you with the capability to specify the library context to be used. You can also use this same function to specify libssl specific properties to use.

In this first example we create two SSL\_CTX objects using two different library contexts.

/\*

\* We assume that a non-default library context with the FIPS provider loaded has been

\* created called fips\_libctx.

/

SSL\_CTX \*fips\_ssl\_ctx = SSL\_CTX\_new\_with\_libctx(fips\_libctx, NULL, TLS\_method());

/\*

\* We assume that a non-default library context with the default provider loaded has been

\* created called non\_fips\_libctx.

/

SSL\_CTX \*non\_fips\_ssl\_ctx = SSL\_CTX\_new\_with\_libctx(non\_fips\_libctx, NULL, TLS\_method());

In this second example we create two SSL\_CTX objects using different properties to specify FIPS usage:

/\*

\* The "fips=yes" property includes all FIPS approved algorithms as well as serializers from the

\* default provider that are allowed to be used. The NULL below indicates that we are using the

\* default library context.

\*/

SSL\_CTX \*fips\_ssl\_ctx = SSL\_CTX\_new\_with\_libctx(NULL, "fips=yes", TLS\_method());

/\*

\* The "provider!=fips" property allows algorithms from any provider except the FIPS provider

\*/

SSL\_CTX \*non\_fips\_ssl\_ctx = SSL\_CTX\_new\_with\_libctx(NULL, "provider!=fips", TLS\_method());

Note that in the OpenSSL alpha 1 and alpha 2 releases OpenSSL does not automatically detect what signature algorithms are available within the currently loaded providers. If signature algorithms in the default set are not available, then an OpenSSL endpoint will offer them anyway. This could result in a handshake failure if the peer decides to use that signature algorithm. As a workaround until this is implemented applications can set the supported signature algorithms manually using a function such as SSL\_CTX\_set1\_sigalgs\_list() or similar. See the man page [[here](https://www.openssl.org/docs/manmaster/man3/SSL_CTX_set1_sigalgs.html)]

**7.8 Confirming that an algorithm is being provided by the FIPS module**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=28)]

A chain of links needs to be followed to go from an algorithm instance to the provider that implements it. The process is similar for all algorithm, here the example of a digest is used.

1. To go from an *EVP\_MD\_CTX* to an *EVP\_MD*, use the **EVP\_MD\_CTX\_md()** call.
2. To go from the *EVP\_MD* to its *OSSL\_PROVIDER*, use the **EVP\_MD\_provider()** call.
3. To extract the name from the *OSSL\_PROVIDER*, use the **OSSL\_PROVIDER\_name()** call.
4. Finally, use strcmp(3) or printf(3) on the name.

8 Openssl command line application changes[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=29)]

The following additional command line arguments have been added

**-provider\_path** path\_name - Provider load path

**-provider** provider\_name - Provider to load

These options can be used multiple times to load any providers, such as the 'legacy' provider or third party providers. If used then the 'default' provider would also need to be specified if required. The -provider\_path must be specified before the -provider option.

9 STATUS of current development[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=30)]

*[this is a collection of notes, changing as time and alpha / beta releases go]*

The current status of OpenSSL 3.0 is **in development**  
The next status is expected to be **alpha**

**9.1 Known issues**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=31)]

**9.1.1 Building and testing**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=32)]

* Doesn't build and test on all platforms on our watch list. See the list of [platforms](https://wiki.openssl.org/index.php/OpenSSL_3.0#Platforms) below

*To be noted that we can't pretend to build on everything and anything, but there are a number of platforms that we watch, either on our own or with community help and reporting*

**9.1.2 Integration**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=33)]

(these issues are tracked in [a table further down](https://wiki.openssl.org/index.php/OpenSSL_3.0#Provider_implementation_support_in_other_OpenSSL_APIs))

* PKCS#7, CMS, SSL/TLS don't work with asymmetric keys implemented by a provider. There's a temporary hack in place that "downgrades" such keys to work with legacy methods (EVP\_PKEY\_METHOD and EVP\_PKEY\_ASN1\_METHOD)
* CMP/CRMF, PKCS#7, TS, CMS, PKCS#12 and OSSL\_STORE currently have no library context support
* OCSP, PEM, ASN.1 have some very limited library context support
* It is not yet possible to "fetch" a RAND algorithm

**9.1.3 Programming**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=34)]

* EVP\_set\_default\_properties() does not work (see [github #11594](https://github.com/openssl/openssl/issues/11594))

**9.1.4 SSL/TLS**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=35)]

* libssl does not currently detect what signature algorithms are available within the currently loaded providers. Unless explicitly configured differently endpoints will advertise to peers the default list of signature algorithms that are supported - even if those are not available in the currently loaded providers. This could result in handshake failures. As a workaround until this is fixed you should explicitly configure signature algorithms that are consistent with the loaded providers.

**9.2 Platforms**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=36)]

These are platforms that have been observed so far. More will be added.

|  |  |  |  |
| --- | --- | --- | --- |
| **Platform** | **Builds** | **Tests** | **Comment** |
| Linux - x86 / x86\_64 | Yes | Yes |  |
| Linux - s390x | Yes | Yes |  |
| FreeBSD - aarch64 | Yes | Yes | Tested on 13.0-CURRENT |
| FreeBSD - amd64 | Yes | Yes | Tested on 12.1-STABLE and 11.3-STABLE |
| FreeBSD - i386 | Yes | Yes | Had to run ./config no-pic due to lack of CAST PIC support |
| Windows + Visual C - x86 / x86\_64 | Yes | Yes |  |
| MacOS X | Yes | Yes |  |
| OpenVMS - Alpha / Itanium | No | Unknown | New include directories need to be dealt with, and more elegantly than the 1.1.1 kludge |

**9.3 Features**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=37)]

All the core support features are in.

The percentages in the tables below represent the amount of work done to convert legacy implementations to a provider based ones. Algorithms for which the conversion hasn't been completed (or ever started) remain full functional via the legacy code paths.

**9.3.1 Provider implemented operation types**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=38)]

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation type** | **Code completion %** | **Documentation completion %** | **Comment** |
| EVP\_DIGEST | 100% | ?? |  |
| EVP\_CIPHER | 100% | ?? |  |
| EVP\_MAC | 100% | ?? |  |
| EVP\_KDF | 100% | ?? |  |
| EVP\_ASYM\_CIPHER | 100% | ?? |  |
| EVP\_KEYEXCH | 100% | ?? |  |
| EVP\_SIGNATURE | 100% | ?? |  |
| EVP\_KEYMGMT | 95% | 70% | Missing functionality for loading HSM keys |
| OSSL\_SERIALIZER | 50% | 50% | Serializer implemented, deserializer not implemented |
| OSSL\_STORE | 0% | 0% |  |

**9.3.2 Provider implemented ciphers**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=39)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| AES | default, FIPS | 100% | ?? |  |
| ARIA | default | 100% | ?? |  |
| BF | legacy | 100% | ?? |  |
| CAMELLIA | default | 100% | ?? |  |
| CAST | legacy | 100% | ?? |  |
| DES | legacy | 100% | ?? |  |
| DESX | legacy | 100% | ?? |  |
| DES-EDE3 | default, FIPS | 100% | ?? | For FIPS, only DES-EDE3-ECB and DES-EDE3-CBC |
| IDEA | legacy | 100% | ?? |  |
| RC2 | legacy | 100% | ?? |  |
| RC4 | legacy | 100% | ?? |  |
| RC5 | legacy | 100% | ?? |  |
| SEED | legacy | 100% | ?? |  |
| SM4 | default | 100% | ?? |  |

**9.3.3 Provider implemented digests**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=40)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| BLAKE2 | default | 100% | ?? |  |
| SM3 | default | 100% | ?? |  |
| MD2 | legacy | 100% | ?? |  |
| MD4 | legacy | 100% | ?? |  |
| MD5, MD5-SHA1 | default | 100% | ?? | MD5-SHA1 is a TLS special, not otherwise useful |
| MDC2 | legacy | 100% | ?? |  |
| SHA1 | default, FIPS | 100% | ?? |  |
| SHA2 | default, FIPS | 100% | ?? |  |
| SHA3 | default, FIPS | 100% | ?? |  |
| SHAKE | default, FIPS | 100% | ?? | For the FIPS provider, only SHAKE-256 is available, not SHAKE-128. |
| RIPEMD-160 | leagcy | 100% | ?? |  |
| WHIRLPOOL | legacy | 100% | ?? |  |

**9.3.4 Provider implemented MACs**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=41)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| BLAKE2 | default | 100% | ?? |  |
| CMAC | default | 100% | ?? |  |
| GMAC | default, FIPS | 100% | ?? |  |
| HMAC | default, FIPS | 100% | ?? |  |
| KMAC | default | 100% | ?? |  |
| POLY1305 | default | 100% | ?? |  |
| SIPHASH | default | 100% | ?? |  |

**9.3.5 Provider implemented KDFs**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=42)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| HKDF | default, FIPS | 100% | ?? |  |
| KBKDF | default | 100% | ?? |  |
| KRB5KDF | default | 100% | ?? | Kerberos KDF |
| PBKDF2 | default, FIPS | 100% | ?? |  |
| SCRYPT | default | 100% | ?? |  |
| SSKDF | default, FIPS | 100% | ?? |  |
| TLS1-PRF | default, FIPS | 100% | ?? | TLS 1.x PRF is treated as a KDF by OpenSSL |
| X942KDF | default | 100% | ?? |  |
| X963KDF | default | 100% | ?? |  |

**9.3.6 Provider implemented asymmetric key types**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=43)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Key type** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| DH | default, FIPS | 95% | ?? |  |
| DSA | default, FIPS | 100% | ?? |  |
| EC | default, FIPS | 100% | ?? |  |
| ED25519, X25519, ED448, X448 | default, FIPS | 100% | ?? | Vendor affirmed for FIPS, they cannot yet be validated. |
| RSA | default, FIPS | 100% | ?? | RSA-PSS or RSA-OAEP are considered separate key types, although the RSA EVP\_ASYM\_CIPHER and EVP\_SIGNATURE implementations carry some of the corresponding properties. |
| RSA-PSS | default | 100% | ?? |  |
| RSA-OAEP | default | 0% | ?? |  |
| SM2 | default | 0% | ?? |  |

**9.3.7 Provider implemented asymmetric ciphers**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=44)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| RSA | default, FIPS | 80% | ?? |  |
| RSAES-OAEP | default | 80% | ?? |  |

**9.3.8 Provider implemented signature**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=45)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| DSA | default, FIPS | 100% | ?? |  |
| ECDSA | default, FIPS | 100% | ?? |  |
| ED25519, ED448 | default, FIPS | 100% | ?? | In the FIPS provider, these are vendor affirmed. |
| RSA, RSASSA-PSS | default, FIPS | 100% | ?? |  |

**9.3.9 Provider implemented key exchange**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=46)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| DH | default, FIPS | 70% | ?? | We lack support for X9.42 DH, which is needed by CMS |
| ECDH | default, FIPS | 100% | ?? |  |
| X25519, X448 | default, FIPS | 100% | ?? | In the FIPS provider, these are vendor affirmed. |

**9.3.10 Provider implemented serializers / deserializers**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=47)]

**9.3.10.1 Serializers**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=48)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Serializer** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| DH to printable text, DER, PEM | default | 100% | ?? |  |
| DSA to printable text, DER, PEM | default | 100% | ?? |  |
| ED25519 to printable text, DER, PEM | default | 100% | ?? |  |
| ED448 to printable text, DER, PEM | default | 100% | ?? |  |
| EC to printable text, DER, PEM | default | 100% | ?? |  |
| RSA to printable text, DER, PEM | default | 100% | ?? |  |
| RSA-PSS to printable text, DER, PEM | default | 100% | ?? |  |
| RSA-OAEP to printable text, DER, PEM | default | 0% ? | ?? |  |
| SM2 to printable text, DER, PEM | default | 0% ? | ?? |  |
| X25519 to printable text, DER, PEM | default | 100% | ?? |  |
| X448 to printable text, DER, PEM | default | 100% | ?? |  |

**9.3.10.2 Deserializers**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=49)]

TO BE ADDED

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Deserializer** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |

**9.3.11 Provider implemented OSSL\_STORE URI schemes**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=50)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **URI scheme** | **Providers** | **Code completion %** | **Documentation completion %** | **Comment** |
| file: | default (?) | 0% | ?? | This is pending on deserializers |

**9.4 Library Context/Provider implementation support in other OpenSSL APIs**[[edit](https://wiki.openssl.org/index.php?title=OpenSSL_3.0&action=edit&section=51)]

Diverse OpenSSL APIs have been modified and continue to be modified to support provider implementations.

|  |  |  |  |
| --- | --- | --- | --- |
| **API** | **Code completion %** | **Documentation completion %** | **Comment** |
| ASN1 | 5% | 5% |  |
| CMS | 0% | 0% | There are hacks in place that downgrade a key to legacy when used with CMS |
| CMP | ?? | ?? | We need to investigate if we need to change anything |
| CRMF | 5% | 0% |  |
| OCSP | 20% | 20% | All changes needed to pass the libssl test suite have been done. We need to investigate if further changes are required |
| OSSL\_STORE | 0% | 0% |  |
| PEM | 50% | 50% | Integrated with provider serializers for writing out keys and parameters |
| PKCS#7 | 0% | 0% | There are hacks in place that downgrade a key to legacy when used with PKCS#7 |
| PKCS#12 | 0% | 0% |  |
| SSL / TLS | 80% | 100% | There are hacks in place that downgrade a key to legacy in some situations. Some processing happens in libssl that should be moved to a provider. Presence of signature algorithms is not correctly detected |
| TS | 0% | 0% |  |
| X509 | 80% | 80% | All changes needed to pass the libssl test suite have been done. We need to investigate if further changes are required |