# Engines

## Deprecation Note

The ENGINE API was introduced in OpenSSL version 0.9.6 as a low level interface for adding alternative implementations of cryptographic primitives, most notably for integrating hardware crypto devices.

The ENGINE interface has its limitations and it has been superseeded by the PROVIDER API, it is deprecated in OpenSSL version 3.0. The following documentation is retained as an aid for users who need to maintain or support existing ENGINE implementations. Support for new hardware devices or new algorithms should be added via providers, and existing engines should be converted to providers as soon as possible.

## Built-in ENGINE implementations

There are currently built-in ENGINE implementations for the following crypto devices:

- Microsoft CryptoAPI
- VIA Padlock
- nCipher CHIL

In addition, dynamic binding to external ENGINE implementations is now provided by a special ENGINE called "dynamic". See the "DYNAMIC ENGINE" section below for details.

At this stage, a number of things are still needed and are being worked on:

1. Integration of EVP support.
2. Configuration support.
3. Documentation!

## Integration of EVP support

With respect to EVP, this relates to support for ciphers and digests in the ENGINE model so that alternative implementations of existing algorithms/modes (or previously unimplemented ones) can be provided by ENGINE implementations.

## Configuration support

Configuration support currently exists in the ENGINE API itself, in the form of "control commands". These allow an application to expose to the user/admin the set of commands and parameter types a given ENGINE implementation supports, and for an application to directly feed string based input to those ENGINEs, in the form of name-value pairs. This is an extensible way for ENGINEs to define their own "configuration" mechanisms that are specific to a given

ENGINE (eg. for a particular hardware device) but that should be consistent across *all* OpenSSL-based applications when they use that ENGINE. Work is in progress (or at least in planning) for supporting these control commands from the CONF (or NCONF) code so that applications using OpenSSL's existing configuration file format can have ENGINE settings specified in much the same way. Presently however, applications must use the ENGINE API itself to provide such functionality. To see first hand the types of commands available with the various compiled-in ENGINEs (see further down for dynamic ENGINEs), use the "engine" openssl utility with full verbosity, i.e.:

```
openssl engine -vvvv
```

### Documentation

Documentation? Volunteers welcome! The source code is reasonably well self-documenting, but some summaries and usage instructions are needed - moreover, they are needed in the same POD format the existing OpenSSL documentation is provided in. Any complete or incomplete contributions would help make this happen.

## STABILITY & BUG-REPORTS

What already exists is fairly stable as far as it has been tested, but the test base has been a bit small most of the time. For the most part, the vendors of the devices these ENGINEs support have contributed to the development and/or testing of the implementations, and *usually* (with no guarantees) have experience in using the ENGINE support to drive their devices from common OpenSSL-based applications. Bugs and/or inexplicable behaviour in using a specific ENGINE implementation should be sent to the author of that implementation (if it is mentioned in the corresponding C file), and in the case of implementations for commercial hardware devices, also through whatever vendor support channels are available. If none of this is possible, or the problem seems to be something about the ENGINE API itself (ie. not necessarily specific to a particular ENGINE implementation) then you should mail complete details to the relevant OpenSSL mailing list. For a definition of "complete details", refer to the OpenSSL "README" file. As for which list to send it to:

- openssl-users: if you are *using* the ENGINE abstraction, either in an pre-compiled application or in your own application code.
- openssl-dev: if you are discussing problems with OpenSSL source code.

## USAGE

The default "openssl" ENGINE is always chosen when performing crypto operations unless you specify otherwise. You must actively tell the openssl utility

commands to use anything else through a new command line switch called "-engine". Also, if you want to use the ENGINE support in your own code to do something similar, you must likewise explicitly select the ENGINE implementation you want.

Depending on the type of hardware, system, and configuration, "settings" may need to be applied to an ENGINE for it to function as expected/hoped. The recommended way of doing this is for the application to support ENGINE "control commands" so that each ENGINE implementation can provide whatever configuration primitives it might require and the application can allow the user/admin (and thus the hardware vendor's support desk also) to provide any such input directly to the ENGINE implementation. This way, applications do not need to know anything specific to any device, they only need to provide the means to carry such user/admin input through to the ENGINE in question. Ie. this connects *you* (and your helpdesk) to the specific ENGINE implementation (and device), and allows application authors to not get buried in hassle supporting arbitrary devices they know (and care) nothing about.

A new "openssl" utility, "openssl engine", has been added in that allows for testing and examination of ENGINE implementations. Basic usage instructions are available by specifying the "-?" command line switch.

# DYNAMIC ENGINES

The new "dynamic" ENGINE provides a low-overhead way to support ENGINE implementations that aren't pre-compiled and linked into OpenSSL-based applications. This could be because existing compiled-in implementations have known problems and you wish to use a newer version with an existing application. It could equally be because the application (or OpenSSL library) you are using simply doesn't have support for the ENGINE you wish to use, and the ENGINE provider (eg. hardware vendor) is providing you with a self-contained implementation in the form of a shared-library. The other use-case for "dynamic" is with applications that wish to maintain the smallest foot-print possible and so do not link in various ENGINE implementations from OpenSSL, but instead leaves you to provide them, if you want them, in the form of "dynamic"-loadable shared-libraries. It should be possible for hardware vendors to provide their own shared-libraries to support arbitrary hardware to work with applications based on OpenSSL 0.9.7 or later. If you're using an application based on 0.9.7 (or later) and the support you desire is only announced for versions later than the one you need, ask the vendor to backport their ENGINE to the version you need.

## How does "dynamic" work?

The dynamic ENGINE has a special flag in its implementation such that every time application code asks for the 'dynamic' ENGINE, it in fact gets its own

copy of it. As such, multi-threaded code (or code that multiplexes multiple uses of 'dynamic' in a single application in any way at all) does not get confused by 'dynamic' being used to do many independent things. Other ENGINEs typically don't do this so there is only ever 1 ENGINE structure of its type (and reference counts are used to keep order). The dynamic ENGINE itself provides absolutely no cryptographic functionality, and any attempt to "initialise" the ENGINE automatically fails. All it does provide are a few "control commands" that can be used to control how it will load an external ENGINE implementation from a shared-library. To see these control commands, use the command-line;

```
openssl engine -vvvv dynamic
```

The "SO_PATH" control command should be used to identify the shared-library that contains the ENGINE implementation, and "NO_VCHECK" might possibly be useful if there is a minor version conflict and you (or a vendor helpdesk) is convinced you can safely ignore it. "ID" is probably only needed if a shared-library implements multiple ENGINEs, but if you know the engine id you expect to be using, it doesn't hurt to specify it (and this provides a sanity check if nothing else). "LIST_ADD" is only required if you actually wish the loaded ENGINE to be discoverable by application code later on using the ENGINE's "id". For most applications, this isn't necessary - but some application authors may have nifty reasons for using it. The "LOAD" command is the only one that takes no parameters and is the command that uses the settings from any previous commands to actually *load* the shared-library ENGINE implementation. If this command succeeds, the (copy of the) 'dynamic' ENGINE will magically morph into the ENGINE that has been loaded from the shared-library. As such, any control commands supported by the loaded ENGINE could then be executed as per normal. Eg. if ENGINE "foo" is implemented in the shared-library "libfoo.so" and it supports some special control command "CMD_FOO", the following code would load and use it (NB: obviously this code has no error checking);

```
ENGINE *e = ENGINE_by_id("dynamic");
ENGINE_ctrl_cmd_string(e, "SO_PATH", "/lib/libfoo.so", 0);
ENGINE_ctrl_cmd_string(e, "ID", "foo", 0);
ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0);
ENGINE_ctrl_cmd_string(e, "CMD_FOO", "some input data", 0);
```

For testing, the "openssl engine" utility can be useful for this sort of thing. For example the above code excerpt would achieve much the same result as;

```
openssl engine dynamic \
          -pre SO_PATH:/lib/libfoo.so \
          -pre ID:foo \
          -pre LOAD \
          -pre "CMD_FOO:some input data"
```

Or to simply see the list of commands supported by the "foo" ENGINE;

```
openssl engine -vvvv dynamic \
          -pre SO_PATH:/lib/libfoo.so \
          -pre ID:foo \
          -pre LOAD
```

Applications that support the ENGINE API and more specifically, the "control commands" mechanism, will provide some way for you to pass such commands through to ENGINEs. As such, you would select "dynamic" as the ENGINE to use, and the parameters/commands you pass would control the *actual* ENGINE used. Each command is actually a name-value pair and the value can sometimes be omitted (eg. the "LOAD" command). Whilst the syntax demonstrated in "openssl engine" uses a colon to separate the command name from the value, applications may provide their own syntax for making that separation (eg. a win32 registry key-value pair may be used by some applications). The reason for the "-pre" syntax in the "openssl engine" utility is that some commands might be issued to an ENGINE *after* it has been initialised for use. Eg. if an ENGINE implementation requires a smart-card to be inserted during initialisation (or a PIN to be typed, or whatever), there may be a control command you can issue afterwards to "forget" the smart-card so that additional initialisation is no longer possible. In applications such as web-servers, where potentially volatile code may run on the same host system, this may provide some arguable security value. In such a case, the command would be passed to the ENGINE after it has been initialised for use, and so the "-post" switch would be used instead. Applications may provide a different syntax for supporting this distinction, and some may simply not provide it at all ("-pre" is almost always what you're after, in reality).

## How do I build a "dynamic" ENGINE?

This question is trickier - currently OpenSSL bundles various ENGINE implementations that are statically built in, and any application that calls the "ENGINE_load_builtin_engines()" function will automatically have all such ENGINEs available (and occupying memory). Applications that don't call that function have no ENGINEs available like that and would have to use "dynamic" to load any such ENGINE - but on the other hand such applications would only have the memory footprint of any ENGINEs explicitly loaded using user/admin provided control commands. The main advantage of not statically linking ENGINEs and only using "dynamic" for hardware support is that any installation using no "external" ENGINE suffers no unnecessary memory footprint from unused ENGINEs. Likewise, installations that do require an ENGINE incur the overheads from only *that* ENGINE once it has been loaded.

Sounds good? Maybe, but currently building an ENGINE implementation as a shared-library that can be loaded by "dynamic" isn't automated in OpenSSL's build process. It can be done manually quite easily however. Such a shared-library can either be built with any OpenSSL code it needs statically linked in, or it can link dynamically against OpenSSL if OpenSSL itself is built as a shared

library. The instructions are the same in each case, but in the former (statically linked any dependencies on OpenSSL) you must ensure OpenSSL is built with position-independent code ("PIC"). The default OpenSSL compilation may already specify the relevant flags to do this, but you should consult with your compiler documentation if you are in any doubt.

This example will show building the "atalla" ENGINE in the crypto/engine/ directory as a shared-library for use via the "dynamic" ENGINE.

1. "cd" to the crypto/engine/ directory of a pre-compiled OpenSSL source tree.

2. Recompile at least one source file so you can see all the compiler flags (and syntax) being used to build normally. Eg;

   ```
   touch hw_atalla.c ; make
   ```

   will rebuild "hw_atalla.o" using all such flags.

3. Manually enter the same compilation line to compile the "hw_atalla.c" file but with the following two changes;

   - add "-DENGINE_DYNAMIC_SUPPORT" to the command line switches,
   - change the output file from "hw_atalla.o" to something new, eg. "tmp_atalla.o"

4. Link "tmp_atalla.o" into a shared-library using the top-level OpenSSL libraries to resolve any dependencies. The syntax for doing this depends heavily on your system/compiler and is a nightmare known well to anyone who has worked with shared-library portability before. 'gcc' on Linux, for example, would use the following syntax;

   ```
   gcc -shared -o dyn_atalla.so tmp_atalla.o -L../.. -lcrypto
   ```

5. Test your shared library using "openssl engine" as explained in the previous section. Eg. from the top-level directory, you might try

   ```
   apps/openssl engine -vvvv dynamic \
         -pre SO_PATH:./crypto/engine/dyn_atalla.so -pre LOAD
   ```

If the shared-library loads successfully, you will see both "-pre" commands marked as "SUCCESS" and the list of control commands displayed (because of "-vvvv") will be the control commands for the *atalla* ENGINE (ie. *not* the 'dynamic' ENGINE). You can also add the "-t" switch to the utility if you want it to try and initialise the atalla ENGINE for use to test any possible hardware/driver issues.

# PROBLEMS

It seems like the ENGINE part doesn't work too well with CryptoSwift on Win32. A quick test done right before the release showed that trying "openssl speed -engine cswift" generated errors. If the DSO gets enabled, an attempt is made to write at memory address 0x00000002.