Switching to AOCL Crypto

A Poweful and Flexible Cryptographic Library from AOCL

Abhiram S abhiram.s@amd.com

Contents

I Key Terminologies **3**

II Introduction **4**

1 What to use? **6**

1.1 New Application 6

1.2 Already Existing Application 6

1.2.1 OpenSSL 3.x Based Application 6

1.2.2 IPP-CP based Application 6

1.2.3 Other library based Application 7

III Getting Started **8**

1.3 An example C code for encryption using a Cipher AES algorithm 9

IV Appendix **12**

2 Link to Other Documentations **13**

2.1 Github [pdf] 13

2.2 Local [markdown] 13

List of Figures

1.1 Test Image 11

Part I

Key Terminologies

Part II

Introduction

AOCL-Cryptograpy is an alternative to many cryptographic libraries like openssl, intel's IPP-CP, wolfssl etc. Integrating a new library into an already exisiting application can be difficult. Understanding this difficulty for the developers to switch from already existing libraries, we developed wrappers and providers which will help you integrate your application with AOCL-Cryptograpy in a matter of minutes. If your application is already using ether one of (OpenSSL, IPP-CP), you can use our provider or wrapper to test out the performance delivered by AOCL-Cryptography or even integrate it permanently.

Using AOCL-Crypto's Native API is better than using Providers or Wrappers as there are some performance overheads associated with them. Our suggested workflow is to get stated with the Providers or Wrapper interfaces, once convinced with the performance, dedicate effort to move to native API.

Link to other documentations can be found in Appendix

Chapter 1

What to use?

1.1 New Application

If you are developing a new application, its recommended to use AOCL-Crypto's native C-API. One more alternative will be to write for OpenSSL and use the Provider interface, but it will have overheads from OpenSSL hence recommended to use AOCL-Cryptography native API.

Please continue reading Getting Started

1.2 Already Existing Application

If you trying to integrate with already exisiting application, it will take time for you to change from your current library provider to AOCL-Cryptography in one go. Hence we recommend you to use OpenSSL provider (if already using OpenSSL) or (IPP Provider), then rewrite the parts of your code step by step slowly until you replace the dependency with OpenSSL or IPP.

1.2.1 OpenSSL 3.x Based Application

Application based on OpenSSL can easily use AOCL-Crypto by configuring it to use the provider. AOCL-Crypto's OpenSSL provider documentation found here, will provide the necessary steps to configure openssl provider for your application. Taking each module, removing OpenSSL code, and replacing with AOCL-Crypto API will allow you to slowly migrate to AOCL-Cryptography without too much effort.

1.2.2 IPP-CP based Application

Application based on IPP-CP can easily use AOCL-Crypto by configuring it to use the wrapper, IPP-CP provider documentation can be found here. Taking each module, removing IPP-CP code, and replacing with AOCL-Crypto API will allow you to slowly migrate to AOCL-Cryptography without too much effort.

1.2.3 Other library based Application

Other Libraries can be a fork of OpenSSL or IPP-CP, in that case the provider or wrapper interface may still work, its not recommended to use provider or wrapper interface in the perticular situation as it may result in undefined behaviour in the cryptographic application and this can cause security vulnerabilities. Some other libraries like libsodium, libsalt, WolfSSL, MbedTLS etc does not have any provider or wrapper implementation.

To migrate from Other Library to AOCL-Cryptography, you can slowly phase out the code which which calls the Other Library and replace it with AOCL-Cryptography, one disadvantage of this approach is that only the part you have replaced with AOCL-Crypto API will be using AOCL-Crypto hence there is still a dependency to the depreciated crypto library.

Part III

Getting Started

For more info go to doxygen ## Flow of AOCL-Crypto

Life cycle of any algorithm of AOCL-Crypto is divided into 4 steps.

1. Support Check - After creating the necessary data-strutures (alc__info_t), one has to check if it's supported. Calling `alc_error_t err = alcp_<algo>_supported(info)` will return `alc_error_t` which will indicate if support succeded. You can check if the support did indeed succeed by calling `alcp_is_error(err)`, this will return true if support is successful.
2. Context Allocation - `alc_<algo>_handle_t handle` contains context `handle.context`, this context is used for storing information internal to AOCL Crypto. You can allocate the context by involking `handle.context = malloc(alcp_<algo>_context_size(info))`. As this memory is allocated by the application, deallocation has to be handled by the application itself.
3. Request - Requesting a context from AOCL-Crypto will finalize the internal paths required to achieve the requested task. You can request by invoking `alcp_<algo>_request(&info,handle)`.
4. Finish/Finalize - Some algorithms require `finish` and `finalize` but most of them only require `finish`. To `finish` the operation, you can involk `alcp_<algo>_finish(&handle)`, once finished the handle is no longer valid and must be destroyed by deallocating context. Optionally you can also write zeros to the context memory.

Every API mentioned above will return an `alc_error_t` which will let you know if any error occured.

1.3 An example C code for encryption using a Cipher AES algorithm

```
#include <stdio.h>

int main(){

  alc_cipher_info_t cinfo = {
    .ci_type = ALC_CIPHER_TYPE_AES,
    .ci_key_info     = {
        .type    = ALC_KEY_TYPE_SYMMETRIC,
        .fmt     = ALC_KEY_FMT_RAW,
        .key     = key,
        .len     = cKeyLen,
    },
    .ci_algo_info   = {
        .ai_mode = ALC_AES_MODE_CFB,
        .ai_iv   = iv,
    },
  };
```

```
  /* Step 1 Support Phase */
  err = alcp_cipher_supported(&cinfo);
  if (alcp_is_error(err)) {
    printf("Error: Not Supported \n");
    goto out;
  }
  else{
    printf("Support succeeded\n");
  }

  /* Step 2 Context Creation Phase */
  handle->ch_context = malloc(alcp_cipher_context_size(&cinfo));
  // Memory allocation failure checking
  if (handle->ch_context == NULL) {
      printf("Error: Memory Allocation Failed!\n");
      goto out;
  }

  /* Step 3 Request a conext */
  // Request a context with cinfo
  err = alcp_cipher_request(&cinfo, handle);
  if (alcp_is_error(err)) {
      printf("Error: Unable to Request \n");
      goto out;
  }
  else{
    printf("Request Succeeded\n");
    return 0;
  }

  /* Additional Step specific to algorithm */
  err = alcp_cipher_encrypt(handle, plaintext, ciphertext, len, iv);
  if (alcp_is_error(err)) {
      printf("Error: Unable to Encrypt \n");
      alcp_error_str(err, err_buf, err_size);
      printf("%s\n", err_buf);
      return -1;
  }

  /* Step 4 Finish/Finalize */
  alcp_cipher_finish(&handle);
  free(handle.ch_context);

}
```

In the above code plaintext, ciphertext, len, iv are assumed to be declared.

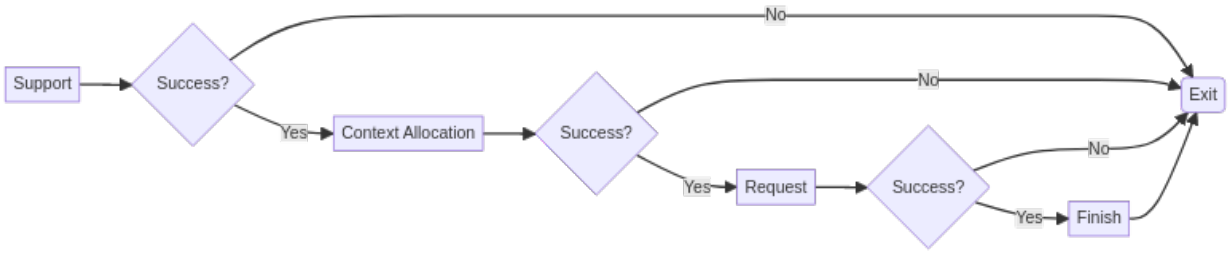

Figure 1.1: Test Image

Part IV

Appendix

Chapter 2

Link to Other Documentations

2.1 Github [pdf]

Latest documentation from Github repo.

1. OpenSSL Provider Documentation
2. IPP Wrapper Documetation
3. AOCL-Crypto API Documentation
4. AOCL Documentation

2.2 Local [markdown]

If viewing as markdown, you can use below links

1. OpenSSL Provider Documentation
2. IPP Wrapper Documetation
3. AOCL-Crypto API Documentation
4. AOCL Documentation
5. AOCL-Crypto Examples