# Switching to AOCL Crypto

A Poweful and Flexible Cryptographic Library from AOCL

Abhiram S abhiram.s@amd.com

# Contents

# List of Figures

# 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. Core Operation - Core Operations of the algorithm involes feeding in the data required by the algorithm. Each algotithm will have its own core operations.
5. 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>
#include <alcp/alcp.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;
}

/* Core Operations 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.

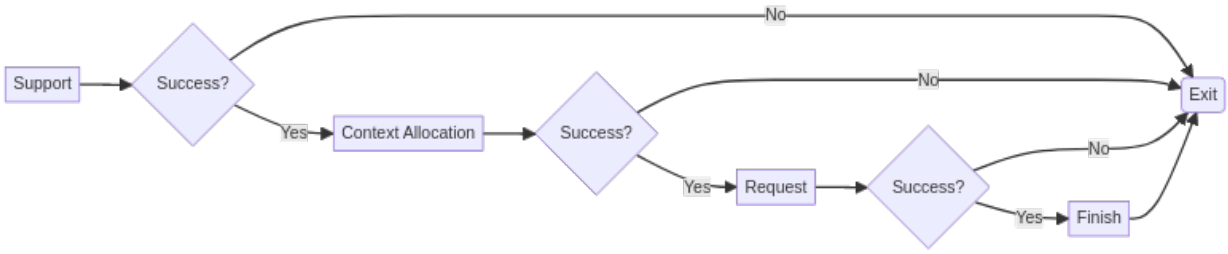


Figure 1.1: Overall Life Cycle Diagram

# Chapter 2

# Symmentric Cipher Algorithms.

There are different ciphers which are supported by ALCP, these ciphers can be grouped into `AEAD` (Authenticated Encrypt Authenticated Decrypt) and `Non AEAD` algorithms. Ciphers which are supported by ALCP are

1. AES
    i. AEAD - Authenticated Encrypt Authenticated Decrypt
        a. GCM
        b. CCM
        c. SIV
    ii. Non AEAD - No authentication data (tag) associated with cipher
        a. CFB - Cipher Feedback.
        b. CBC - Cipher Block Chaining.
        c. CTR - Counter.
        d. XTS - XEX-based tweaked-codebook mode with ciphertext stealing
        e. OFB - Output Feedback.
2. ChaCha
    i. Stream Cipher Mode
    ii. Block Cipher Mode
    iii. Chache20-Poly1305 AEAD

## 2.1 AES (Advanced Encryption Standard)

AES is the defacto standard for internet communication. AES has been time tested and is trusted even for military applications.

## 2.2 Chacha (Modified Salsa with better security)

Chacha has been chosen to succeed AES as the standard for internet communication. In the present most of the Certificate Authorities set ChaCha20 as the default encryption scheme.

## 2.3 API Usage AEAD

Life cycle of AEAD cipher should be all similar. AES, Chacha20-Poly all should use same API for AEAD. Life cycle of different modes of encryption or decryption will differ. This is because each AEAD comes with its own way of handling data.

All AEAD API has this format `alcp_cipher_aead_<operation>`. Any API which starts with `alcp_cipher_aead_` can be considered as an AEAD API.

Life Cycle of AEAD Core Operations

1. Setting IV - Every AEAD except SIV expects an Initialization vector to be set. For SIV this step is skipped. For CCM mode, it requires tag length to be set prior to IV.
2. Setting AAD - `Additional Data` is the `Authentication Data`. This data will influence the final tag generation. It will serve to improve the authenticity of message as any difference in this authentication data will result in an entirely different tag.
3. Encrypt/Decrypt - A plaintext or ciphertext can be given to the algorithm at this stage. Plaintext will be encrypted into ciphertext whereas ciphertext will be decrypted into plaintext.
4. Tag Generation - Once a tag is generated, we can concude that the transaction is complete. Now only `finish` call will be possible on the current state. For Decryption this tag must be compared with the expected tag to make sure that the plaintext is authentic.

### 2.3.1 SIV

Conceptual example not a working example of SIV operation.

Example Code for AES-SIV,

```
// Info about the cipher operation
alc_cipher_aead_info_t cinfo = {
    .ci_type = ALC_CIPHER_TYPE_AES,
    .ci_algo_info   = {
        .ai_mode = ALC_AES_MODE_SIV,
        .ai_iv   = NULL,
        .ai_siv.xi_ctr_key = &kinfo,
    },
    /* No padding, Not Implemented yet*/
    //.pad     = ALC_CIPHER_PADDING_NONE,
    .ci_key_info     = {
```

```
        .type    = ALC_KEY_TYPE_SYMMETRIC,
        .fmt     = ALC_KEY_FMT_RAW,
        .key     = key_cmac,
        .len     = key_len,
    },
};

// Support Check
err = alcp_cipher_aead_supported(&cinfo);
if (alcp_is_error(err)) {
    printf("Error: not supported \n");
}
printf("supported succeeded\n");
/*
  * Application is expected to allocate for context
  */

// Context Allocation
handle.ch_context = malloc(alcp_cipher_aead_context_size(&cinfo));

// Request
err = alcp_cipher_aead_request(&cinfo, &handle);
if (alcp_is_error(err)) {
    printf("Error: unable to request \n");
    alcp_error_str(err, err_buf, err_size);
}
printf("request succeeded\n");

/* For SIV directly step 2 as IV is generated synthetically */
err = alcp_cipher_aead_set_aad(&handle, aad, aad_len);
if (alcp_is_error(err)) {
    printf("Error: unable to encrypt \n");
    alcp_error_str(err, err_buf, err_size);
    return false;
}

/* Step 3 Encrypt stage, tag gets generated here */
// Memory for IV can be memory for tag, its unused by the API
err = alcp_cipher_aead_encrypt(&handle, plaintxt, ciphertxt, len, iv);
if (alcp_is_error(err)) {
    printf("Error: unable to encrypt \n");
    alcp_error_str(err, err_buf, err_size);
    return false;
}
```

```
/* Step 4 Tag Generation */
// Tag in this case generated will be the synthetic IV
err = alcp_cipher_aead_get_tag(&handle, iv, 16);
if (alcp_is_error(err)) {
    printf("Error: unable to encrypt \n");
    alcp_error_str(err, err_buf, err_size);
    return false;
}

// Finish the operation
alcp_cipher_aead_finish(&handle);

// Deallocate context
free(handle.context);
```

### 2.3.2 CCM

Conceptual example not a working example of CCM operation.

Example Code for AES-CCM,

```
// Info about the cipher operation
alc_cipher_aead_info_t cinfo = {
    .ci_type = ALC_CIPHER_TYPE_AES,
    .ci_algo_info   = {
        .ai_mode = ALC_AES_MODE_CCM,
        .ai_iv   = iv,
    },
    /* No padding, Not Implemented yet*/
    //.pad     = ALC_CIPHER_PADDING_NONE,
    .ci_key_info     = {
        .type    = ALC_KEY_TYPE_SYMMETRIC,
        .fmt     = ALC_KEY_FMT_RAW,
        .key     = key,
        .len     = key_len,
    },
};

// Support Check
err = alcp_cipher_aead_supported(&cinfo);
if (alcp_is_error(err)) {
    printf("Error: not supported \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
```

```
}
printf("supported succeeded\n");

// Context Allocation
handle.ch_context = malloc(alcp_cipher_aead_context_size(&cinfo));
if (!handle.ch_context)
    return -1;

// Request
err = alcp_cipher_aead_request(&cinfo, &handle);
if (alcp_is_error(err)) {
    printf("Error: unable to request \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}
printf("request succeeded\n");

/* Step 0 Additional step for CCM, set tag length */
err = alcp_cipher_aead_set_tag_length(&handle, tagLen);
if (alcp_is_error(err)) {
    printf("Error: unable getting tag \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

/* Step 1 set IV */
err = alcp_cipher_aead_set_iv(&handle, ivLen, iv);
if (alcp_is_error(err)) {
    printf("Error: unable ccm encrypt init \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

/* Step 2 set Additional Data */
err = alcp_cipher_aead_set_aad(&handle, ad, adLen);
if (alcp_is_error(err)) {
    printf("Error: unable ccm add data processing \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

/* Step 3 Encrypt/Decrypt */
err =
    alcp_cipher_aead_encrypt_update(&handle, plaintxt, ciphertxt, len, iv);
```

```
if (alcp_is_error(err)) {
    printf("Error: unable encrypt \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

/* Step 4 Get the Tag */
err = alcp_cipher_aead_get_tag(&handle, tag, tagLen);
if (alcp_is_error(err)) {
    printf("Error: unable getting tag \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

// Finish
alcp_cipher_aead_finish(&handle);

// Deallocate the context
free(handle.ch_context);
```

### 2.3.3 GCM

Conceptual example not a working example of GCM operation.

Example Code for AES-GCM,

```
// Info about the cipher operation
alc_cipher_aead_info_t cinfo = {
    .ci_type = ALC_CIPHER_TYPE_AES,
    .ci_algo_info   = {
        .ai_mode = ALC_AES_MODE_GCM,
        .ai_iv   = iv,
    },
    /* No padding, Not Implemented yet*/
    //.pad     = ALC_CIPHER_PADDING_NONE,
    .ci_key_info     = {
        .type    = ALC_KEY_TYPE_SYMMETRIC,
        .fmt     = ALC_KEY_FMT_RAW,
        .key     = key,
        .len     = key_len,
    },
};

// Support Check
err = alcp_cipher_aead_supported(&cinfo);
```

```
if (alcp_is_error(err)) {
    printf("Error: not supported \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

// Context Allocation
handle.ch_context = malloc(alcp_cipher_aead_context_size(&cinfo));
if (!handle.ch_context)
    return -1;

// Request
err = alcp_cipher_aead_request(&cinfo, &handle);
if (alcp_is_error(err)) {
    printf("Error: unable to request \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

/* Step 1 Set IV */
err = alcp_cipher_aead_set_iv(&handle, ivLen, iv);
if (alcp_is_error(err)) {
    printf("Error: unable gcm encrypt init \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

/* Step 2 set additional data */
err = alcp_cipher_aead_set_aad(&handle, ad, adLen);
if (alcp_is_error(err)) {
    printf("Error: unable gcm add data processing \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

/* Step 3 Encrypt or Decrypt */
err =
    alcp_cipher_aead_encrypt_update(&handle, plaintxt, ciphertxt, len, iv);
if (alcp_is_error(err)) {
    printf("Error: unable encrypt \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}
```

```
/* Step 4 Get Tag */
err = alcp_cipher_aead_get_tag(&handle, tag, tagLen);
if (alcp_is_error(err)) {
    printf("Error: unable getting tag \n");
    alcp_error_str(err, err_buf, err_size);
    return -1;
}

// Finish the operation
alcp_cipher_aead_finish(&handle);

// Deallocate the context
free(handle.context)
```

# Part IV

# Appendix

# Chapter 3

# Link to Other Documentations

## 3.1 Github [pdf]

Latest documentation from Github repo.

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

## 3.2 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