CypherShield SDK API Reference Guide

Table of Contents

1 Introduction

The CypherShield Encryption SDK provides a client side encryption technology which can be embedded into customer projects and provide end-to-end quantum resistant security. This can be deployed encapsulated inside TLS sessions, providing quantum save transports inside current protocol standards.

This technology can also be deployed natively adding post-quantum security to environments including wireless device technologies, core carrier transports, and a variety of infrastructures.

1.1 Technical Overview

The CypherShield technologies provide the following security features

a. Client-side encryption library to extend customer technologies or enhance service offerings
b. Symmetric Key Exchange key management. Existing Key-Exchange technologies using PIK are vulnerable to quantum attacks. Symmetric Key Exchange technologies are vulnerable to encryption key exposures using out-of-band key exchanges.The CypherShield technologies provide in-band quantum-resistant key exchanges.
c. Envelope Key Encapsulation – These technologies use current Envelop Key Encapsulation techniques, which allow CypherEye customers to use and secure their data using secret keys, and the CypherShield technologies provides a secure transport, securing data from technology and infrastructure providers.
d. Quantum Secure Encapsulation – CypherShield technologies provide leading edge quantum secure algorithms providing future-proof security and privacy from quantum technologies.
e. Digital Signatures and Authentication – Advanced data signing and end device authentication protect from Man-in-the-Middle spoofing, attacks and other vulnerabilities across public network transports.

2 Keys and Certificates

The CypherShield Encryption SDK provides a mechanism to insure the client-to-client security realm. This realm is defined by the customer using public and private security keys and certificates.

These keys must be included in the customer implementations.

The generation of these keys is defined in the CypherShield Encryption SDK Quick Start Guide.

3 Attributes

The following API attributes are used in SDK library functions.

3.1.1 nA and nB

nA and nB are pre-defined device or end-session names used to identify and authenticate end-to-end traffic. These names are generated by the customer systems and may be end-user names, endpoint device names, etc.

These names must be known by both the sending and receiving application. This is usually administered by organizational directories. Examples include LDAP or Active Directory systems, or community directories like messaging address books.

3.1.2 Key Certification URL

Every client system must have a DNS address for the central CypherShield Key Certification server. This server must be reachable by every client and provides end-to-end authentication for every encryption session.

3.1.3 CypherShield Public Key

The CypherShield provides utilities to generate unique customer public and private keys. The public key must be embedded into each customer client product. This key provides a secure realm for communications between customer clients as well as secure communications to the CypherShield Key Certification server.

3.1.4 Protocol Version

The CypherShield protocol is version aware. This version number must be included in the communications initializations.

Future versions of this product will remain consistent with prior versions using this value.

3.1.5 Keystore Password

CypherShield Encryption SDK provides facilities to store local keys in a private encrypted key store. The key to this store is created and maintained by the customer depending on the implementation.

4 Endpoint APIs

4.1 asInitialize

Initialize environment for the:

- CypherShield Key Certification Server
- Local key storage
- The CypherShield public key
- Other protocol parameters like protocol version, etc.

4.1.1 Example Usage

```
    // Authentication server URL
    std::string url = "jeff-is-a-dork.cyphereye-test.com";
    // Authentication protocol version (currently v1.0 is the latest)
    std::string proto = "v1.0";
    // a server public key
    std::string pubkey =
"9E0DD336376959B661505B008EFD7FBBAE89B9EA8CBB6D2CC0585AF3239E4AA622570F
6A50D157C4F0FABE4E70CE3115C74415B6784FDBC20CCD0C774A525420B04900001E930
00051BD00009B0601007399000087A30000CAB6000022F10000A60A0000F01C0000E714
0000A54B0000C5280100CA3B01092090000A20800004A4900005124000B50A0000F10
80000BB87000053510000E59D00002F7D000040EA00009A960000E0490000F240000097
AC00001E470000AC950100A3CA00007D250000C8A30000BE6700006E8F0000BDA90000E
E96000097BA0000647E0000BC2E000076BE00004A570000B495000013AD0000EFE20000
24420000182A000037A70000A86E0000409B0000B0CE0000EC75000077C70000AE44000
```

```
042590000D8A600008FA30000739F0000699000001B9B0000656B00000EC20000C0AC00
00";
asInitialize(url, proto, pubkey, dirpath, dirvers, aeskey,
    [&]
        result = 0;
        printf("asInitialize()\n");
    },
    [&] (int errCode, std::string errorMessage) {
        // write error to log
        printf("asInitialize() -> Error: %d \n", errCode);
        result = errCode;
    });

    if (result == 0) {
        // Initialized with default values
        printf("asInitialize() executed\n");
    };

    printf("result=%d\n", result);
    if (result != 0) exit(-1);
```

4.2 asInitChannels

The Key Certificate server maintains a channel session between each sender and receiver client. On session startup, the server records client name string of each client, and maintains a Channel session.

During session standup, the Key Certificate server guarantees the end-point identity of each client during the communications session.

4.2.1 Example Usage

The following example, a unique sender and receiver name is created using the system clock.

```
    // sender name base
    std::string nA = "";
    // recipient name base
    std::string nB = "";
    char stime[12];
    sprintf(stime, "%d", (int) time(NULL));
    // each run a name shall be different
    std::string randtime = "";
    randtime += stime;
    // sender name
    nA = "AA"+ randtime;
    // recipient name
    nB = "BB"+ randtime;
    asInitChannels(
    [&] () {
        printf("asInitChannels()\n");
    },
    [&] (int errCode, std::string errorMessage) {
        // write error to log
        printf("asInitChannels() -> Error: %d \n", errCode);
     });
```

4.3 asEncryptMessage

asEncryptMessage performs

- the proprietary quantum-resistant 3-pass key exchange between the sender and receiver client
- Encrypts the payload using the default AES256 encryption algorithms
- Transfers the encrypted payload to the receiver system

4.3.1 Example Usage

```
// Sender creates a local session key and encrypts a message
// a text to send
std::string text_src = "hello world";
// where encrypted text will be stored
std::string text_cpt = "";
asEncryptMessage(text_src, nA, nB,
[&] (std::string _nA, std::string _nB) {
    printf("  onkeychange()\n");
},
[&] (std::string messageOut) {
    // saving encrypted text
        text_cpt = messageOut;
    printf("  asEncryptMessage()\n");
},
[&] (int errCode, std::string errorMessage) {
    // write error to log
    printf("  asEncryptMessage() -> Error: %d \n", errCode);
});
```

4.4 asDecryptMessage

asDecryptMessage is used on the receiver system and using the 3-pass exchanged key, decrypts the network payload, and passes the data to the customer program.

4.4.1 Example Usage

```
// where decrypted text will be stored
std::string text_dst = "";
asDecryptMessage(text_cpt, nA, nB,
[&] (std::string _nA, std::string _nB) {
    printf("  onkeychange()\n");
},
[&] (std::string messageOut) {
    // saving the message
    text_dst = messageOut;
    printf("  asDecryptMessage()\n");
},
[&] (int errCode, std::string errorMessage) {
    // write error to log
    printf("  asDecryptMessage() -> Error: %d \n", errCode);
});
```

4.5 asResetSession

asResetSession provides a mechanism for the customer application to invalidate the current encryption session. This action will trigger the network receiver system to initiate a new 3-pass key-exchange.

The purpose of this is to provide the customer a means to invalidate sessions and trigger new key exchanges. This guarantees any brute force security attacks or snooping attempts will be thwarted.

4.5.1 Example Usage

```
// regenerate a session (on a sender side)
asResetSession(nA, nB,
[&] (std::string _nA, std::string _nB) {
    // the session key was erased
    // recipient session key will be received with the next message
    printf("  onkeychange()\n");
},
[&] () {
    printf("  asResetSession()\n");
},
[&] (int errCode, std::string errorMessage) {
    // write error to log
    printf("  asResetSession() -> Error: %d \n", errCode);
});

// source and destination message comparison
if (text_src == text_dst) {
    printf("  Message crypt/decrypt: OK\n");
} else {
    printf("  Message crypt/decrypt: ERROR\n");
};
```

4.6 asGetChannelKeys

asGetChannelKeys provides a mechanism to retrieve the local keys used on the Key Certificate Server.

4.6.1 Example Usage

```
// Receive locally stored session keys
asGetChannelKeys(nA, nB,
[&] (std::string _result, std::string _nA, std::string _nB) {
    // save keys in a variable
    text_src = _result;
    printf("  as_GetChannelKeys()\n");
},
[&] (int errCode, std::string errorMessage) {
    // write error to log
    printf("  as_GetChannelKeys() -> Error: %d \n", errCode);
});

// Print a key
//console.log(keys_src);

// Realloc memory for string
std::string keys_tmp = text_src;
```

4.7 asSetChannelKeys

asSetChannelKeys provides a mechanism to store Key Certificate channel keys directly. This function is handled automatically with higher level functions.

4.7.1 Example Usage

```
// Load previously stored keys to memory
asSetChannelKeys(keys_tmp, nA, nB,
[&] (std::string _nA, std::string _nB) {
    // save keys in a variable
    printf("  as_SetChannelKeys()\n");
},
[&] (int errCode, std::string errorMessage) {
    // write error to log
    printf("  as_SetChannelKeys() -> Error: %d \n", errCode);
});
```

4.8 asGetChannels

asGetChannels retrieves all local session channels from the local system.

4.8.1 Example Usage

```
// Reading keys again to verify
asGetChannelKeys(nA, nB,
[&] (std::string _result, std::string _nA, std::string _nB) {
    text_dst = _result;
    printf("  as_GetChannelKeys()\n");
},
[&] (int errCode, std::string errorMessage) {
    // write error to log
    printf("  as_GetChannelKeys() -> Error: %d \n", errCode);
});
```

4.9 asSetChannels

asSetChannels initializes the local client session channel storage DB.

4.9.1 Example Usage

```
// Initialize DB back
asSetChannels(items_tmp,
[&] () {
    printf("  as_SetChannels()\n");
},
[&] (int errCode, std::string errorMessage) {
    // write error to log
    printf("  as_SetChannels() -> Error: %d \n", errCode);
});
```

4.10 asDoneChannels

asDoneChannels destroys the session channel storage and deallocates local memory.

4.10.1 Example Usage

```
// Releasing used memory
asDoneChannels(
[&] () {
    printf("as_DoneChannels()\n");
},
[&] (int errCode, std::string errorMessage) {
    printf("as_DoneChannels() -> Error: %d \n", errCode);
});
```

5 Example Programs

Example programs using these API functions are provided in the SDK distribution package inside the Example Programs folder.

The following platforms are supported:

- Linux Native
- Node.js
- Android
- iOS