



Data Encryption User Guide

support@agora.io



[June 2016](#)

Contents

Introduction	3
Implementing Your Customized Data Encryption Algorithm	4
Data Encryption Process	4
Enabling Data Encryption	5
<i>Registering a Packet Observer</i>	5
<i>Implementing a Customized Data Encryption Algorithm</i>	6
<i>Registering Instance</i>	8
Windows	8
Android	8
iOS/Mac	9
Using Agora SDK Built-in Encryption	10
Data Encryption Process	10
Enabling Data Encryption	10

Introduction

Agora SDKs allow your application to encrypt audio and video packets in one of the following ways:

-  [Implementing Your Customized Data Encryption Algorithm](#)
-  [Using the Built-in Encryption of Agora SDK](#)

Note: For descriptions of all APIs mentioned in this document, refer to each platform's reference manual.

Implementing Your Customized Data Encryption Algorithm

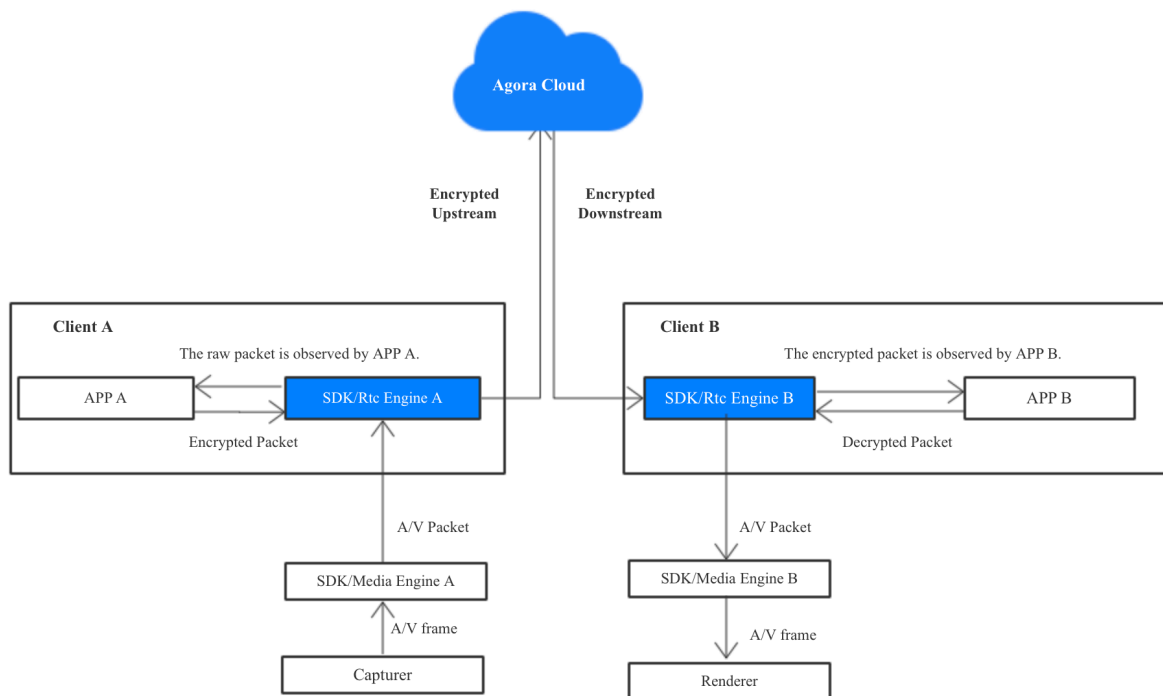
Agora SDKs allow your application to encrypt audio and video packets by implementing the customized data encryption algorithm of your application.

These are the supported SDKs:

- 🔗 Agora Native SDK for Windows
- 🔗 Agora Native SDK for Mac
- 🔗 Agora Native SDK for iOS
- 🔗 Agora Native SDK for Android

Data Encryption Process

The following diagram below depicts the data encryption/decryption process:



Enabling Data Encryption

These are the steps:

1. [Registering a Packet Observer](#)
2. [Implementing Customized Data Encryption Algorithm](#)
3. [Registering Instance](#)

Registering a Packet Observer

Agora Native SDK allows your application to register a packet observer to receive events whenever an audio or video packet is transmitting.

Register a packet observer on your application using the following API:

```
virtual int registerPacketObserver(IPacketObserver* observer);
```

The observer must inherit from `agora::rtc::IPacketObserver` and be implemented in C++. The following is the definition of the `IPacketObserver` class:

```
class IPacketObserver
{
public:

    struct Packet
    {
        const unsigned char* buffer;
        unsigned int size;
    };
    /**
     * called by sdk before the audio packet is sent to other participants
     * @param [in,out] packet:
     *     buffer *buffer points the data to be sent
     *     size of buffer data to be sent
     * @return returns true to send out the packet, returns false to
     discard the packet
     */
    virtual bool onSendAudioPacket(Packet& packet) = 0;
    /**
     * called by sdk before the video packet is sent to other participants
     * @param [in,out] packet:
     *     buffer *buffer points the data to be sent
     *     size of buffer data to be sent
     * @return returns true to send out the packet, returns false to
     discard the packet
     */
}
```

```

virtual bool onSendVideoPacket(Packet& packet) = 0;
/**
 * called by sdk when the audio packet is received from other
participants
 * @param [in,out] packet
 *      buffer *buffer points the data to be sent
 *      size of buffer data to be sent
 * @return returns true to process the packet, returns false to discard
the packet
 */
virtual bool onReceiveAudioPacket(Packet& packet) = 0;
/**
 * called by sdk when the video packet is received from other
participants
 * @param [in,out] packet
 *      buffer *buffer points the data to be sent
 *      size of buffer data to be sent
 * @return returns true to process the packet, returns false to discard
the packet
 */
virtual bool onReceiveVideoPacket(Packet& packet) = 0;

```

Implementing a Customized Data Encryption Algorithm

Inherit from `agora::rtc::IPacketObserver` to implement the customized data encryption algorithm on your application.

The following example uses XOR for data processing. For Agora Native SDK, sending and receiving packets are handled by the different threads, which is why encryption and decryption can use different buffers:

```

class AgoraPacketObserver : public agora::rtc::IPacketObserver
{
public:
    AgoraPacketObserver()
    {
        m_txAudioBuffer.resize(2048);
        m_rxAudioBuffer.resize(2048);
        m_txVideoBuffer.resize(2048);
        m_rxVideoBuffer.resize(2048);
    }
    virtual bool onSendAudioPacket(Packet& packet)
    {
        int i;
        //encrypt the packet
        const unsigned char* p = packet.buffer;
        const unsigned char* pe = packet.buffer+packet.size;

```

```

    for (i = 0; p < pe && i < m_txAudioBuffer.size(); ++p, ++i)
    {
        m_txAudioBuffer[i] = *p ^ 0x55;
    }
    //assign new buffer and the length back to SDK
    packet.buffer = &m_txAudioBuffer[0];
    packet.size = i;
    return true;
}

virtual bool onSendVideoPacket(Packet& packet)
{
    int i;
    //encrypt the packet
    const unsigned char* p = packet.buffer;
    const unsigned char* pe = packet.buffer+packet.size;
    for (i = 0; p < pe && i < m_txVideoBuffer.size(); ++p, ++i)
    {
        m_txVideoBuffer[i] = *p ^ 0x55;
    }
    //assign new buffer and the length back to SDK
    packet.buffer = &m_txVideoBuffer[0];
    packet.size = i;
    return true;
}

virtual bool onReceiveAudioPacket(Packet& packet)
{
    int i = 0;
    //decrypt the packet
    const unsigned char* p = packet.buffer;
    const unsigned char* pe = packet.buffer+packet.size;
    for (i = 0; p < pe && i < m_rxAudioBuffer.size(); ++p, ++i)
    {
        m_rxAudioBuffer[i] = *p ^ 0x55;
    }
    //assign new buffer and the length back to SDK
    packet.buffer = &m_rxAudioBuffer[0];
    packet.size = i;
    return true;
}

virtual bool onReceiveVideoPacket(Packet& packet)
{
    int i = 0;
    //decrypt the packet
    const unsigned char* p = packet.buffer;
    const unsigned char* pe = packet.buffer+packet.size;

```

```

    for (i = 0; p < pe && i < m_rxVideoBuffer.size(); ++p, ++i)
    {
        m_rxVideoBuffer[i] = *p ^ 0x55;
    }
    //assign new buffer and the length back to SDK
    packet.buffer = &m_rxVideoBuffer[0];
    packet.size = i;
    return true;
}

private:
    std::vector<unsigned char> m_txAudioBuffer; //buffer for sending audio
data
    std::vector<unsigned char> m_txVideoBuffer; //buffer for sending video
data

    std::vector<unsigned char> m_rxAudioBuffer; //buffer for receiving
audio data
    std::vector<unsigned char> m_rxVideoBuffer; //buffer for receiving
video data
};

```

Registering Instance

Windows

Call **registerPacketObserver** to register the instance of the `agora::rtc::IPacketObserver` class implemented by your application.

Android

1. Implement a Java wrapper.

For example,

```

JNIEXPORT jint JNICALL
Java_io_agora_video_demo_RtcEngineEncryption_enableEncryption(JNIEnv
*env, jclass clazz, jlong engineHandle)
{
    typedef jint (*PFN_registerAgoraPacketObserver)(void* engine,
agora::rtc::IPacketObserver* observer);

    void* handle = dlopen("libagora-rtc-sdk-jni.so", RTLD_LAZY);
    if (!handle)
    {
        __android_log_print(ANDROID_LOG_ERROR, "agora encrypt demo",

```



```

    "cannot find libagora-rtc-sdk-jni.so");
    return -1;
}

PFN_registerAgoraPacketObserver pfn =
(PFN_registerAgoraPacketObserver)dlsym(handle,
"registerAgoraPacketObserver");

if (!pfn)
{
    __android_log_print(ANDROID_LOG_ERROR, "aogra encrypt demo",
"cannot find registerAgoraPacketObserver");
    return -2;
}

return pfn((void*)engineHandle, &s_packetObserver);
}

```

Java wrapper:

```

public class RtcEngineEncryption {
    static {
        System.loadLibrary("agora-encrypt-demo-jni");
    }
    public static native int enableEncryption(long rtcEngineHandle);
}

```

2. Call **registerPacketObserver** to register the instance of `agora::rtc::IPacketObserver` class implemented by your application:

iOS/Mac

Call **registerAgoraPacketObserver** to register the instance of `agora::rtc::IPacketObserver` class implemented by your application.

Using Agora SDK Built-in Encryption

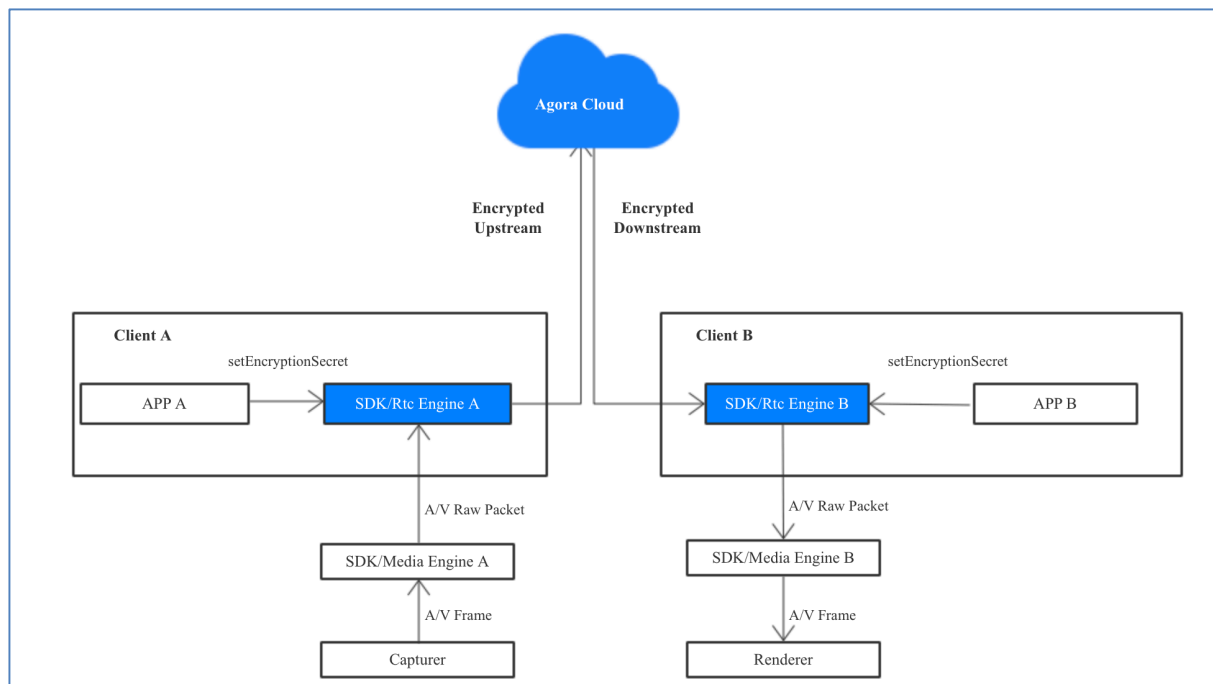
Agora SDKs support built-in encryption using the AES-128 encryption algorithm.

These SDKs are currently available:

- Agora Native SDK for Windows
- Agora Native SDK for Mac
- Agora Native SDK for iOS
- Agora Native SDK for Android
- Agora Native SDK for Web

Data Encryption Process

The following diagram depicts the built-in encryption/decryption process:



Enabling Data Encryption

Agora Native SDK introduces a new API `setEncryptionSecret` to enable and set the encryption key. The SDK generates a 128-bit encryption key for communication based on the secret string and channel name. Since the encryption key is generated by both channel name and secret, it is different even when the same secret is used in different channels.

The encryption key is dynamically generated according to the secret specified by your application, so ensure that all participants in the same channel set the same secret. Agora Native SDK only uses the secret on the client locally and does not transmit the secret to the server side.