ALIBABA CLOUD

阿里云

IoT设备身份认证最佳实践

文档版本：20201221

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.使用ID²-KM Demo试用ID²

本文介绍如何使用ID²-KM Demo试用ID²。

ID²-KM Demo可实现基于ID²的应用和业务的前期调试。使用时ID²密钥预置在固件中，不需要烧录ID²和HAL接口的对接。

方案一：在第三方OS中使用ID²-KM Demo

1. 获取ID²-KM Demo源码。
2. 参考在Link Kit SDK上适配ID²-KM进行设备端适配。

   > **说明** 不需要烧录ID²和对接HAL接口。

3. 登录物联网设备身份认证获取调试类ID²。

   > **说明** 调试 ID²的密钥类型选择AES。

4. 预置调试类ID²。

   在km_demo.c（irot/demo）中导入获取的调试ID²。

   ```
   #define ID2_ID      "000FFFE[redacted]0"
   #define ID2_KEY     "e[redacted]b332736990598a452ffe33a97"

   #define ID2_LEN                         24
   #define MAX_KEY_LEN                     32
   ```

   > **说明** 预制密钥ID2_KEY必须是hex-string格式。

5. 生成调试固件。

   i. 重新编译生成KM Demo库 `libkm.a` 。

   ii. 使用ID²-KM Demo库替换系统中的KM库，重新编译生成固件。

方案二：在AliOS Things中使用ID²-KM Demo

1. 登录物联网设备身份认证获取调试类ID²。

   > **说明** 调试 ID²的密钥类型选择AES。

2. 预置调试ID²。

   在km_demo.c（secrutiy/irot/demo）中导入获取的调试ID²。

   ```
   #define ID2_ID      "000FFFE[redacted]0"
   #define ID2_KEY     "e[redacted]b332736990598a452ffe33a97"

   #define ID2_LEN                         24
   #define MAX_KEY_LEN                     32
   ```

   > **说明** 预制密钥ID2_KEY必须是hex-string格式。

3. 生成调试固件。

   i. 在 `security/irot/aos.mk` 中配置载体为libkm_demo。

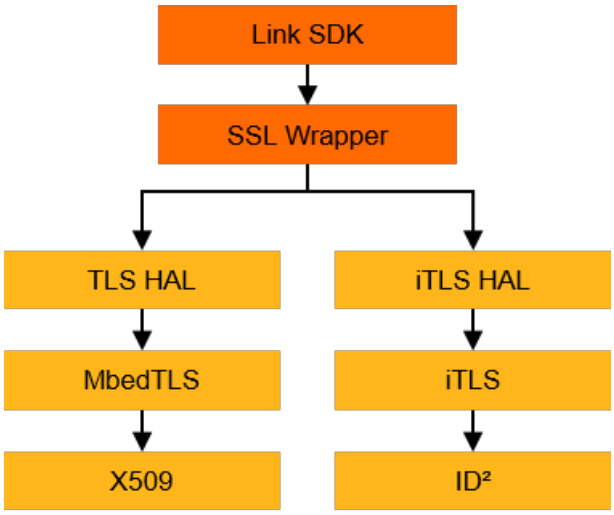
```
NAME := irot

$(NAME)_MBINS_TYPE := kernel
$(NAME)_VERSION := 2.0.0
$(NAME)_SUMMARY := root of trust, based on mcu, se or tee
#if component's header files under another directory, add RPM_INCLUDE_DIR to indicate where the header file folder is located
RPM_INCLUDE_DIR := ../include/irot

GLOBAL_INCLUDES += ../include/irot

ifeq ($(CONFIG_LS_KM_SE), y)
$(NAME)_COMPONENTS := libkm_se
else ifeq ($(CONFIG_LS_KM_TEE), y)
$(NAME)_COMPONENTS := libkm_tee
else
$(NAME)_COMPONENTS := libkm_demo
endif
```

   ii. 重新编译生成调试固件。

2.Link SDK v3.x 集成ID²

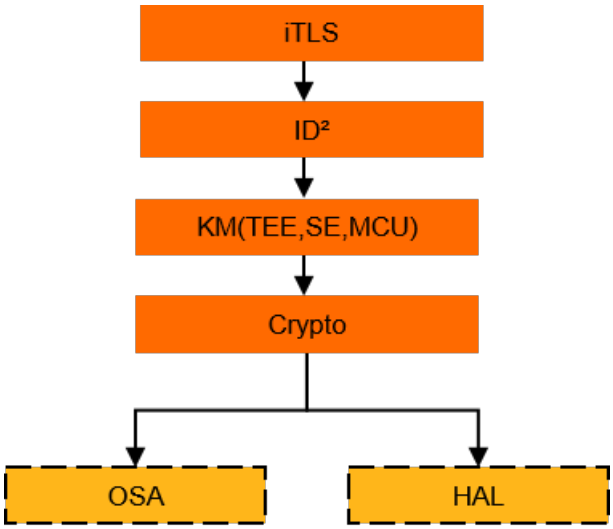
本文介绍Link SDK v3.x 集成ID²的操作步骤和示例代码的运行过程。

Link SDK v3.x 集成ID²原理如下图所示。



操作步骤

1. 基于Link SDK代码抽取机制，完成SDK功能代码的抽取和HAL接口适配。
2. 移植安全连接协议iTLS和ID²至目标平台。其中，OSA和HAL需要厂商根据接口进行适配；其他模块，由厂商提供编译工具，阿里云IoT进行适配。移植全流程如下图。



3. 适配Link SDK的HAL_Crypt_xxx。适配过程中可调用ID²中的Crypto模块，示例代码如下：

```
#include "infra_compat.h"
#include "ali_crypto.h"
#define AES_BLOCK_SIZE 16
#define KEY_LEN 16 // aes 128 cbc
#define p_aes128_t p_HAL_Aes128_t
#define PLATFORM_AES_ENCRYPTION HAL_AES_ENCRYPTION
#define PLATFORM_AES_DECRYPTION HAL_AES_DECRYPTION
```

```
#define PLATFORM_AES_DECRYPTION HAL_AES_DECRYPTION
void *HAL_Malloc(uint32_t size);
void HAL_Free(void *ptr);
p_HAL_Aes128_t HAL_Aes128_Init(
        const uint8_t *key,
        const uint8_t *iv,
        AES_DIR_t dir)
{
    ali_crypto_result result;
    void *          aes_ctx;
    size_t          aes_ctx_size, alloc_siz;
    uint8_t *       p;
    bool            is_en = true; // encrypto by default
    if (dir == PLATFORM_AES_DECRYPTION) {
        is_en = false;
    }
    result = ali_aes_get_ctx_size(AES_CBC, &aes_ctx_size);
    if (result != ALI_CRYPTO_SUCCESS) {
        HAL_Printf("get ctx size fail(%08x)", result);
        return NULL;
    }
    alloc_siz = aes_ctx_size + KEY_LEN * 2 + sizeof(bool);
    aes_ctx  = HAL_Malloc(alloc_siz);
    if (aes_ctx == NULL) {
        HAL_Printf("kmalloc(%d) fail", (int)aes_ctx_size);
        return NULL;
    }
    memset(aes_ctx, 0, alloc_siz);
    p = (uint8_t *)aes_ctx + aes_ctx_size;
    memcpy(p, key, KEY_LEN);
    p += KEY_LEN;
    memcpy(p, iv, KEY_LEN);
    p += KEY_LEN;
    *((bool *)p) = is_en;
    return aes_ctx;
}
int HAL_Aes128_Destroy(p_HAL_Aes128_t aes)
{
    if (aes) {
        HAL_Free(aes);
    }
```

```
    return 0;
}
static int platform_aes128_encrypt_decrypt(p_HAL_Aes128_t aes_ctx,
                                           const void *src, size_t siz,
                                           void *dst, aes_type_t t)
{
    ali_crypto_result result;
    size_t         dlen, in_len = siz, ctx_siz;
    uint8_t *      p, *key, *iv;
    bool           is_en;
    if (aes_ctx == NULL) {
        HAL_Printf("platform_aes128_encrypt_decrypt aes_ctx is NULL");
        return -1;
    }
    result = ali_aes_get_ctx_size(AES_CBC, &ctx_siz);
    if (result != ALI_CRYPTO_SUCCESS) {
        HAL_Printf("get ctx size fail(%08x)", result);
        return 0;
    }
    p   = (uint8_t *)aes_ctx + ctx_siz;
    key = p;
    p += KEY_LEN;
    iv = p;
    p += KEY_LEN;
    is_en = *((uint8_t *)p);
    in_len <<= t == AES_CBC ? 4 : 0;
    dlen = in_len;
    result = ali_aes_init(t, is_en, key, NULL, KEY_LEN, iv, aes_ctx);
    if (result != ALI_CRYPTO_SUCCESS) {
        HAL_Printf("ali_aes_init fail(%08x)", result);
        return 0;
    }
    result = ali_aes_finish(src, in_len, dst, &dlen, SYM_NOPAD, aes_ctx);
    if (result != ALI_CRYPTO_SUCCESS) {
        HAL_Printf("aes_finish fail(%08x)", result);
        return -1;
    }
    return 0;
}
int HAL_Aes128_Cbc_Encrypt(
        p_HAL_Aes128_t aes,
```

```
        const void *src,
        size_t blockNum,
        void *dst)
{
  return platform_aes128_encrypt_decrypt(aes, src, blockNum, dst, AES_CBC);
}
int HAL_Aes128_Cbc_Decrypt(
        p_HAL_Aes128_t aes,
        const void *src,
        size_t blockNum,
        void *dst)
{
  return platform_aes128_encrypt_decrypt(aes, src, blockNum, dst, AES_CBC);
}
int HAL_Aes128_Cfb_Encrypt(
        p_HAL_Aes128_t aes,
        const void *src,
        size_t length,
        void *dst)
{
  return platform_aes128_encrypt_decrypt(aes, src, length, dst, AES_CFB128);
}
int HAL_Aes128_Cfb_Decrypt(
        _IN_ p_HAL_Aes128_t aes,
        _IN_ const void *src,
        _IN_ size_t length,
        _OU_ void *dst)
{
  return platform_aes128_encrypt_decrypt(aes, src, length, dst, AES_CFB128);
}
```

4. 适配Link SDK的SSL HAL。
   i. 添加HAL_TLS_itls.c和HAL_DTLS_itls.c至eng/wrappers/tls目录。
   ii. 删除eng/wrappers/tls目录下的HAL_TLS_mbedtls.c和HAL_DTLS_mbedtls.c文件。
5. 集成Link SDK ID²库。
   i. 添加头文件和静态库。

      添加ID²相关头文件到eng/wrappers/include目录。

      添加ID²相关静态库release/lib目录。

ii. 修改编译脚本Makefile。

链接ID²相关的静态库。

```
CFLAGS  += -DSUPPORT_ITLS
LDFLAGS += -L./release/lib -litls -lid2 -lkm -licrypt -lls_hal -lls_osa
```

移除默认mbedtls模块的链接。

```
WRAPPER_IMPL_C := $(shell find $(SRCDIR) -name "*.c" -path "*wrappers*" -not -path "*mbedtls*")
```

运行示例代码

1. 在 wrappers/os/xxx/HAL_OS_xxx.c 中设置参数值，如下所示。

```
char _product_key[IOTX_PRODUCT_KEY_LEN + 1]     = "a1MZxOd****";
char _product_secret[IOTX_PRODUCT_SECRET_LEN + 1] = "h4I4dneEFp7E****";
char _device_name[IOTX_DEVICE_NAME_LEN + 1]     = "test_01";
char _device_secret[IOTX_DEVICE_SECRET_LEN + 1]  = "t9G******";
```

> **说明** product_key和product_secret在产品创建（选择开通ID²服务）时生成；device_name设备编号，使用ID²服务时，由厂商自行确定，需保证产品维度内的唯一性；device_secret设备密钥，使用ID²服务时，可设置为任意合法的字符串。

2. 在样例程序中，配置ID²服务。

以examples/mqtt_example.c为例。

```
/**
 *
 * MQTT connect hostname string
 *
 * MQTT server's hostname can be customized here
 *
 * default value is ${productKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com
 */
/* mqtt_params.host = "something.iot-as-mqtt.cn-shanghai.aliyuncs.com"; */
char host[64] = {0};
HAL_Snprintf(host, 64, "%s.itls.cn-shanghai.aliyuncs.com", DEMO_PRODUCT_KEY);
mqtt_params.host = host;
mqtt_params.port = 1883;
mqtt_params.customize_info = "authtype=id2";
```

3. 在SDK根目录，执行make命令，完成样例程序的编译。

```
make clean
make
```

4. 在已烧录ID²的设备上运行样例程序mqtt example。确认运行日志如下。

```
[inf] Connecting to /a1WO4Z9****.itls.cn-shanghai.aliyuncs.com/1883...
[inf] ok
[inf] . Setting up the SSL/TLS structure...
[inf] ok
<LS_LOG> ID2 Client Version: 0x00020000
<LS_LOG> ID2 Client Build Time: Sep 26 2019 15:29:43
<LS_LOG> -----------------------------------------
<LS_LOG> CONFIG_ID2_DEBUG is not defined!
<LS_LOG> CONFIG_ID2_OTP is defined!
<LS_LOG> CONFIG_ID2_KEY_TYPE: ID2_KEY_TYPE_AES
<LS_LOG> -----------------------------------------
[inf] Performing the SSL/TLS handshake...
<LS_LOG> id2_client_get_id 655: ID2: 00FFFFFFFFFFFF20A1C****
<LS_LOG> mbedtls_parse_auth_code_ext 407: . Verify iTLS Server authCode OK!
[inf] ok
....
<{
<  "message": "hello!"
<}
example_message_arrive|031 :: Message Arrived:
example_message_arrive|032 :: Topic : /a1WO4Z9****/example1/user/get
example_message_arrive|033 :: Payload: {"message":"hello!"}
```

ID²相关调试

- iTLS建连失败时，首先通过查看消息警告（alert message）进行问题排查。

```
[inf] Connecting to /a1WO4Z9****.itls.cn-shanghai.aliyuncs.com/1883...
[inf]  ok
[inf]  . Setting up the SSL/TLS structure...
[inf]  ok
<LS_LOG> ID2 Client Version: 0x00020000
<LS_LOG> ID2 Client Build Time: Sep 26 2019 15:29:43
<LS_LOG> ----------------------------------------
<LS_LOG> CONFIG_ID2_DEBUG is not defined!
<LS_LOG> CONFIG_ID2_OTP is defined!
<LS_LOG> CONFIG_ID2_KEY_TYPE: ID2_KEY_TYPE_AES
<LS_LOG> ----------------------------------------
[inf] Performing the SSL/TLS handshake...
<LS_LOG> id2_client_get_id 655: ID2: 00FFFFFFFFFFFA6F67A****
<LS_LOG> mbedtls_ssl_handle_message_type 4194: got an alert message, type: [2:162]
[err] failed ! mbedtls_ssl_handshake returned -0x7780
```

上述样例日志中，2表示FATAL类型的警告，162表示消息警告类型。iTLS常见的错误警告类型如下表：

错误码	错误信息
0	Peer close notify
10	Unexpected message
20	Bad record mac
50	Decode error
51	Decryption error
110	Unsupported extension
160	ID2 generic error
161	ID2 no quota
162	ID2 is not exist
163	ID2 authcode is invalid
164	ID2 has not been activated
165	The timestamp used in authcode is expired
166	ID2 challenge is invalid
167	Not support this operation

错误码	错误信息
168	ID2 has been suspended
169	ID2 has been discarded
170	Permission denied, id2 has been blinded to other product key
171	Product key is invalid
172	Product key is not exist
173	ID2 server is busy
174	The device fingerprint is invalid
175	The device fingerprint is duplicated
176	ID2 server random is invalid
177	Hash type used in authcode generated is invalid
178	ID2 key type is invalid

- 若通过查看消息警告不能确定错误原因，可使用iTLS和ID²调试版本进行调试。
  i. 使用iTLS和ID²调试库替换SDK中的正式库。
  ii. 在HAL_xx_itls.c中配置调试级别宏（DEBUG_LEVEL）。

```
/*< set debug log level, 0 close*/
#define DEBUG_LEVEL    0
```

上述样例中0表示No debug，日志关闭，只有极少错误信息输出。

DEBUG_LEVEL参数说明如下：

参数值	描述
0	No debug，日志关闭，只有极少错误信息输出。
1	Error，错误日志。
2	State change，错误日志和状态日志。
3	Informational，错误日志、状态日志和调试日志。
4	Verbose，输出所有日志。