阿里云
IoT设备身份认证

最佳实践

文档版本：20200323

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. **阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	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。
	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。
	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。	执行`cd /d C:/window`命令，进入Windows系统文件夹。
*## *	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 ID²试用指南-ID²-KM Demo

本文介绍了如何试用ID²。

> **说明:**
> **ID²-KM Demo 是指通过ID²密钥在固件中预置的方式（不需要烧录ID²和HAL接口的对接。），实现基于ID²的应用和业务的前期调试。**

方案一、在第三方OS中使用Demo KM

1，获取Demo KM源码：

ID²安全SDK v2.0源码： *https://github.com/alibaba/id2_client_sdk*

2，设备端适配：

参考链接： *https://help.aliyun.com/document_detail/142889.html*

> **说明:**
> **不需要烧录ID²、不需要进行HAL接口的对接。**

3，获取调试类ID²：

登录*ID²*控制台

> **说明:**
> **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：

```
10  #define ID2_ID      "000FFFF[illegible]0"
11  #define ID2_KEY     "e4[illegible]b332736990598a452ffe33a97"
12
13  #define ID2_LEN                         24
14  #define MAX_KEY_LEN                     32
15
```

5，生成调试固件：

1）重新编译生成KM Demo库（libkm.a）。

2）使用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Demo KM:

1，获取调试ID²：

登录ID²控制台

说明:

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：

```
10  #define ID2_ID      "000FFFF[illegible]0"
11  #define ID2_KEY     "e4ca[illegible]b332736990598a452ffe33a97"
12
13  #define ID2_LEN                         24
14  #define MAX_KEY_LEN                     32
15
```

3，生成调试固件

1) 在security/irot/aos.mk, 配置载体为Demo KM:

```
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
```

2）重新编译生成调试固件。

2 设备端SDK 适配手册

此手册适用于：

- **硬件厂商、物联网开发者需要为某一款硬件/设备适配ID²。**
- **设备使用了Android，Linux，AliOS Things系统。**
- **使用了任一种安全载体：SE、SIM、KM、TEE。**
- **如果您的硬件/设备已经完成了ID²与设备端底层硬件的适配，请直接跳过此文档；请直接对接ID²的接口。ID²设备端接口说明参见：设备端对接 *API***

设备端安全SDK是打包集成了阿里云IoT在设备端的安全框架和安全组件，通过统一的OSA和HAL接口，以便适配到不同的系统和平台。目前已经支持Android，Linux，AliOS Things，如果厂商希望在更多的设备类型上集成和使用这套安全SDK，请与我们联系。

一、OS适配接口：

基础功能：

1，void ls_osa_print(const char *fmt, …)

- **功能：打印函数，用于向串口或其他标准输出打印日志或调试信息。**
- **参数：**

名称	数据类型	描述
fmt	const char *	格式化字符串
…	void *	可变参数列表

2，int ls_osa_snprintf(char *str, size_t size, const char *fmt, ...)

- **功能：打印函数，向内存缓冲区格式化构建一个字符串。**
- **参数：**

名称	数据类型	描述
str	char *	指向字符串缓冲的指针
size	size_t	缓冲区的长度
fmt	const char *	格式化字符串
…	void *	可变参数列表

- **返回值：实际写入缓冲区的字符串长度。**

3，void *ls_osa_malloc(size_t size)

- 功能：申请一块堆内存。
- 参数：

名称	数据类型	描述
size	size_t	申请内存的字节大小

- 返回值：指向申请内存首地址的指针，失败返回NULL。

4，void *ls_osa_calloc(size_t nmemb, size_t size)

- 功能：分配nmemb个长度为size的连续堆内存，且内存数据置为0。
- 参数：

名称	数据类型	描述
nmemb	size_t	内存块的数量
size	size_t	单个内存的字节长度

- 返回值:指向申请内存首地址的指针，失败返回NULL。

5，void ls_osa_free(void *ptr)

- 功能: 释放参数ptr指向的一块堆内存。
- 参数：

名称	数据类型	描述
ptr	void *	指向要释放的堆内存的地址

6，void ls_osa_msleep(unsigned int msec)

- 功能：睡眠函数，使当前执行线程睡眠指定的毫秒数。
- 参数：

名称	数据类型	描述
msec	unsigned int	线程挂起的时间，单位毫秒

7，long long ls_osa_get_time_ms(void)

- 功能：获取当前系统的时间戳大小。
- 参数：void
- 返回值：系统的时间戳大小（以毫秒为单位）。

多线程:

1，int ls_osa_mutex_create(void **mutex)

- **功能: 创建一个互斥量，用于多线程下的同步访问。**
- **参数:**

名称	数据类型	描述
mutex	void **	指向创建互斥量的句柄

- **返回值：成功：= 0**

失败：< 0

2，void ls_osa_mutex_destroy(void *mutex)

- **功能：销毁互斥量的句柄。**
- **参数:**

名称	数据类型	描述
mutex	void *	互斥量的句柄

3，int ls_osa_mutex_lock(void *mutex)

- **功能：锁住一个互斥量。**
- **参数:**

名称	数据类型	描述
mutex	void *	互斥量的句柄

- **返回值:**

成功：= 0失败：< 0

4，int ls_osa_mutex_unlock(void *mutex)

- **功能：解锁一个互斥量。**
- **参数:**

名称	数据类型	描述
mutex	void *	互斥量的句柄

- **返回值:**

成功：= 0失败：< 0

网络操作:

1，int ls_osa_net_connect(const char *host, const char *port, int type)

- **功能：创建由host:port指定的特定类型的网络连接。**
- **参数：**

名称	数据类型	描述
host	const char *	连接的主机地址
port	const char*	连接的端口号
type	int	网络类型，LS_NET_TYPE_XXX

- **返回值：**

成功：网络连接的句柄失败：-1

2，void ls_osa_net_disconnect(int fd)

- **功能：断开网络连接，并释放相应资源。**
- **参数：**

名称	数据类型	描述
fd	int	网络连接的句柄

3，int ls_osa_net_send(int fd, unsigned char *buf, size_t len, int *ret_orig)

- **功能：发送数据到指定的网络中。**
- **参数：**

名称	数据类型	描述
fd	int	网络连接的句柄
buf	unsigned char*	发送数据的内存
len	size_t	发送数据的字节长度
ret_orig	int *	接口返回失败时，指向具体的错误码

- **返回值：**

成功：实际发送数据的长度失败：-1

4，int ls_osa_net_recv(int fd, unsigned char *buf, size_t len, int timeout, int * ret_orig)

- **功能：在指定的时间内，从网络中读取最多len字节的数据。**
- **参数：**

名称	数据类型	描述
fd	int	网络连接的句柄
buf	unsigned char*	接收数据的内存
len	size_t	内存的最大长度
timeout	int	读取数据的时间值，0代码阻塞
ret_orig	int *	接口返回失败时，指向具体的错误码

- **返回值：**

成功：实际接收数据的长度失败：-1

二、HAL适配接口

Soft-KM：

1，int ls_hal_get_dev_id(uint8_t *dev_id, uint32_t *id_len)

- **功能：获取设备唯一ID。**
- **参数：**

名称	数据类型	描述
dev_id	uint8_t *	存储设备唯一ID的内存
id_len	uint32_t *	内存的长度（in），设备ID的实际长度(out)。如果输入*id_len小于实际的device id 的长度， 将*id_len置为device id的实际长度，同时返回-1。

- **返回值：**

成功：= 0失败：-1

参考实现如下：

```
#define DEV_ID "12345678"
```

```
#define DEV_ID_LEN 8

int ls_hal_get_dev_id(uint8_t *dev_id, uint32_t *id_len)
{
    int ret = 0;

    if (*id_len < DEV_ID_LEN) {
        if (*id_len != 0) {
            ls_osa_print("short buffer id len is %d\n", *id_len);
        }
        *id_len = DEV_ID_LEN;
        return -1;
    }

    if (!dev_id) {
        return -1;
    }

    memcpy(dev_id, DEV_ID, DEV_ID_LEN);
    *id_len = DEV_ID_LEN;

    return ret;
}
```

2，int ls_hal_open_rsvd_part(int flag)

- **功能：根据指定的权限（flag）打开预留的管理分区；如不支持文件系统，直接返回0。**

> **说明:**
> **预留的管理分区：密钥存储，至少2K。正常使用和OTA 升级时，数据不被擦除。**

- **参数：**

名称	数据类型	描述
flag	int	flag: LS_HAL_READ: 只读打开 LS_HAL_WRITE：只写打开 LS_HAL_READ | LS_HAL_WRITE：读写打开

- **返回值：**

成功：文件句柄失败：-1

3，int ls_hal_write_rsvd_part(int fd, uint32_t offset, void *data, uint32_t data_len)

- **功能：向指定的分区中，写入data_len字节的数据。**
- **参数：**

名称	数据类型	描述
fd	int	文件句柄；或者忽略（没有文件系统）
offset	uint32_t	写数据的偏移量
data	void *	写入的数据
data_len	data_len	写入数据的长度（字节）

· 返回值：

成功：= 0失败：-1

4，int ls_hal_read_rsvd_part(int fd, uint32_t offset, void *buffer, uint32_t read_len)

· 功能：从指定的分区中，读取read_len字节的数据。

· 参数：

名称	数据类型	描述
fd	int	文件句柄；或者忽略（没有文件系统）
offset	uint32_t	读数据的偏移量
buffer	void *	读取数据的缓存
read_len	data_len	读取数据的长度（字节）

· 返回值：

成功：= 0失败：-1

5，int ls_hal_close_rsvd_part(int fd)

· 功能：关闭打开的预留分区。

· 参数：

名称	数据类型	描述
fd	int	文件句柄；或者忽略（没有文件系统）

· 返回值：

成功：= 0失败：-1

Secure Storage:

1，int ls_hal_kv_init(void)

- **功能：初始化安全存储模块。**
- **参数：void**
- **返回值：**

成功：= 0失败：< 0

2，void ls_hal_kv_deinit(void)

- **功能：注销安全存储模块。**
- **参数：void**

3，int ls_hal_kv_set(const char *key, const void *value, int len, int sync)

- **功能：设置一组key-value到安全存储中。**
- **参数：**

名称	数据类型	描述
key	const char *	存储数据的标识
value	const void *	存储的数据
len	int	存储数据的长度（字节）
sync	int	同步/异步

- **返回值：**

成功：= 0失败：< 0

4，int ls_hal_kv_get(const char *key, void *buffer, int *buffer_len)

- **功能：根据标识（key），从安全存储中获取对应的数据。**
- **参数：**

名称	数据类型	描述
key	const char *	存储数据的标识
buffer	void *	存储数据的缓存
buffer_len	int *	缓存长度/读取的数据长度

- **返回值：**

成功：= 0失败：< 0

5，int ls_hal_kv_del(const char *key)

- **功能：删除安全存储中key标识的数据。**
- **参数：**

名称	数据类型	描述
key	const char *	存储数据的标识

- **返回值：**

成功：= 0失败：< 0

三、适配说明

OS适配：

在AliOS Things中，已经完成OS的适配；在第三方OS上，可根据应用/场景选择相应的OS接口适配，其中基础功能（必须），多线程（多线程场景）和网络操作（使用iTLS安全连接）。

平台配置：

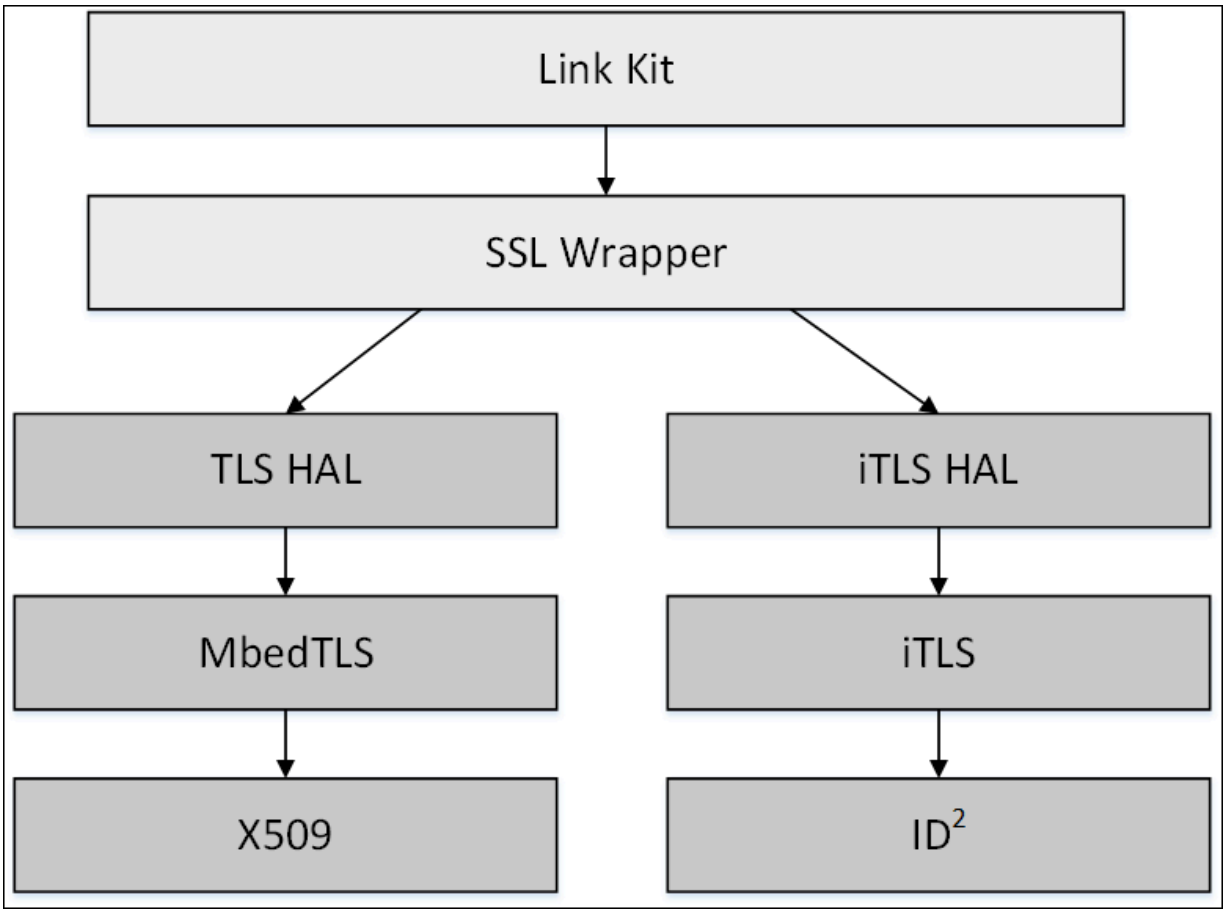
AliOS Things 2.1.0版本开始，Link Security SDK的配置统一提取到平台的aos.mk中，如mk3080(board/mk3080/aos.mk):

```
GLOBAL_DEFINES += CLI_CONFIG_STACK_SIZE=4096

# Link Security Config
CONFIG_LS_DEBUG      := n
CONFIG_LS_ID2_OTP    := y
CONFIG_LS_KM_SE      := n
CONFIG_LS_KM_TEE     := n

CONFIG_SYSINFO_PRODUCT_MODEL := ALI_AOS_MK3080
CONFIG_SYSINFO_DEVICE_NAME   := MK3080

GLOBAL_CFLAGS += -DSYSINFO_PRODUCT_MODEL=\"$(CONFIG_SYSINFO_PRODUCT_MODEL)\"
GLOBAL_CFLAGS += -DSYSINFO_DEVICE_NAME=\"$(CONFIG_SYSINFO_DEVICE_NAME)\"

GLOBAL_CFLAGS += -L $($(NAME)_LOCATION)

# Extra build target include bootloader, and copy output file to eclipse debug file (copy_o
```

其中：CONFIG_LS_DEBUG: 控制模块调试信息的开启和关闭。CONFIG_LS_ID2_OTP：控制ID2空发的开启和关闭，其中SE不支持空发，配置无效。CONFIG_LS_KM_SE: 控制SE KM的开启和关闭。CONFIG_LS_KM_TEE: 控制TEE KM的开启和关闭。

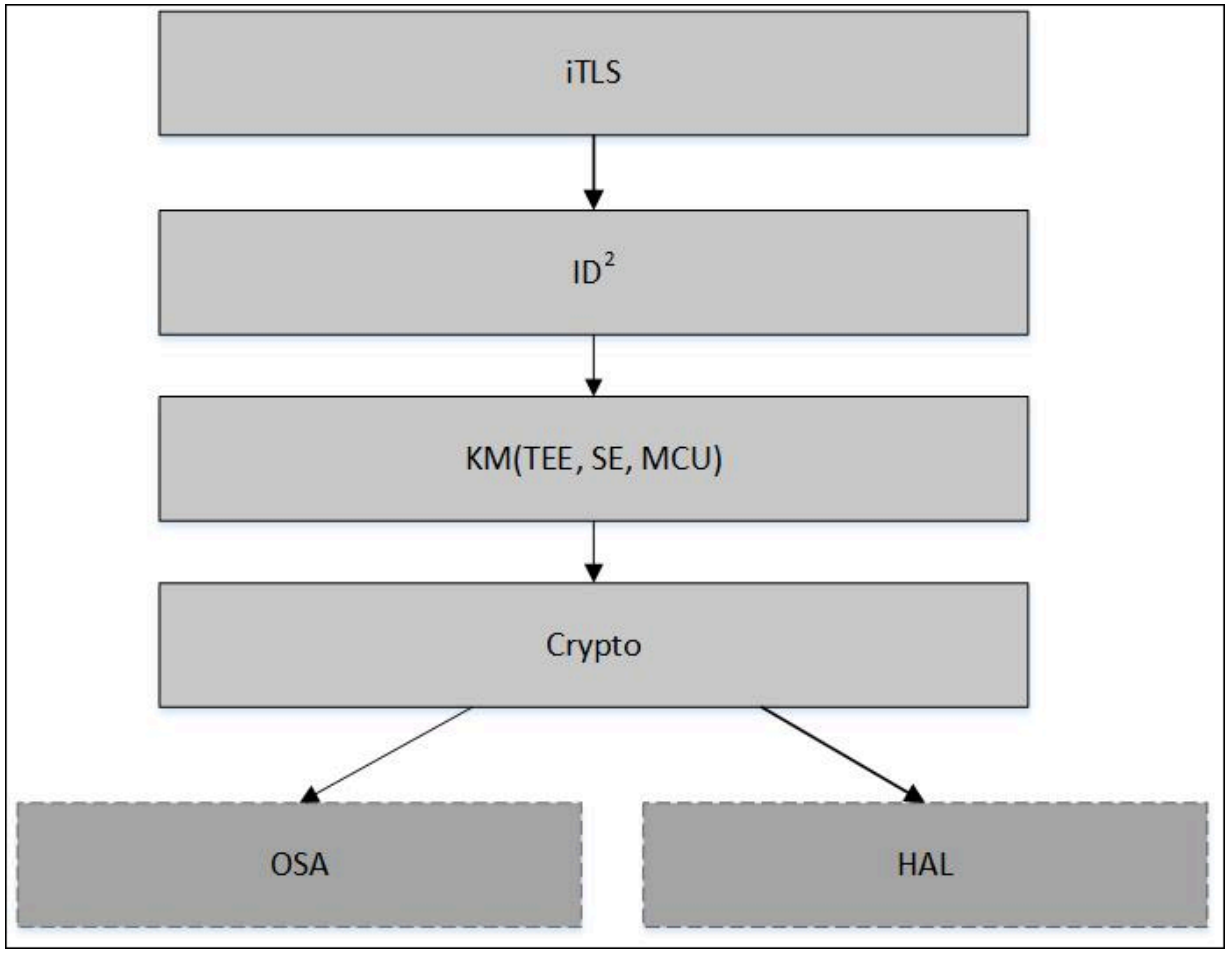
3 Link Kit v3.x 集成ID²

IoT设备身份认证ID²（Internet Device ID），是一种物联网设备的可信身份标识，具备不可篡改、不可伪造、全球唯一的安全属性；轻量安全连接协议（iTLS），基于ID²进行双向身份认证，提供与标准TLS相同的安全能力，同时减少协议对设备和网络的依赖，满足物联网（IoT）对连接安全的需求。



1，基于Link Kit代码抽取机制，完成SDK功能代码的抽取和HAL接口适配。

2，要实现安全连接协议（iTLS）和ID²在目标平台的移植，OSA和HAL需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ID²团队进行适配。



3，Link Kit“HAL_Crypt_”文件的适配可调用ID²中的Crypto模块，参考实现如下：

```
 #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

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，Link Kit SSL HAL的适配：

1）添加HAL_TLS_itls.c和HAL_DTLS_itls.c到eng/wrappers/tls目录。

2）删除eng/wrappers/tls目录下的HAL_TLS_mbedtls.c和HAL_DTLS_mbedtls.c文件。

5，Link Kit ID²库集成：

1）添加头文件和静态库：

```
A. 添加ID²相关头文件到eng/wrappers/include目录。
B. 添加ID²相关静态库release/lib目录。
```

2）修改编译脚本Makefile：

```
A. 链接ID²相关的静态库：
CFLAGS  += -DSUPPORT_ITLS
LDFLAGS += -L./release/lib -litls -lid2 -lkm -licrypt -lls_hal -
lls_osa

B. 移除默认mbedtls模块的链接：
WRAPPER_IMPL_C := $(shell find $(SRCDIR) -name "*.c" -path "*wrappers
*" -not -path "*mbedtls*")
```

6，样例程序演示：

1）填写设备证书到例程（wrappers/os/xxx/HAL_OS_xxx.c）中：

```
char _product_key[IOTX_PRODUCT_KEY_LEN + 1]       = "a1M******";
char _product_secret[IOTX_PRODUCT_SECRET_LEN + 1] = "h4I******";
char _device_name[IOTX_DEVICE_NAME_LEN + 1]       = "tes******";
char _device_secret[IOTX_DEVICE_SECRET_LEN + 1]   = "t9******";
```

说明：

product_key和product_secret在产品创建（选择开通ID²服务）时生成。 device_name为设备名称，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，配置使用ID²服务：

如examples/mqtt_example.c：

```
     /**
      *
      *  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";
```

7，样例程序演示：

1）在SDK顶层目录运行如下命令。

```
make clean
make
```

2）在已烧录ID²的设备上运行mqtt example，得到如下日志。

```
 [inf] Connecting to /a1WO4Z9qHRw.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: 00F******
<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  : /a1WO4Z9qHRw/example1/user/get
example_message_arrive|033 :: Payload: {"message":"hello!"}
```

8，ID²相关调试：

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

```
 [inf] Connecting to /a1WO4Z9qHRw.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: 00F******
```

```
<LS_LOG> mbedtls_ssl_handle_message_type 4194: got an alert message,
type: [2:162]
[err] failed  ! mbedtls_ssl_handshake returned -0x7780
```

说明:

iTLS常见的错误警告参见文档：*iTLS*常见错误码。

2）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。

```
A. 使用iTLS和ID²调试库替换SDK中的正式库。
B. 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
```

其中：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，所有日志输出。
```

4 等保2.0（物联网）对接说明

本文介绍了如何通过阿里云IoT安全提供的产品和服务，满足等保2.0二级、三级的物联网扩展部分要求的安全措施。

1，等保2.0（物联网）安全套餐

等保2.0（物联网）安全套餐的方案能够覆盖等保2.0物联网扩展要求的控制点：

等保2.0（物联网扩展）	阿里云 等保2.0物联网安全套餐 覆盖的控制层面和控制点	套餐购买链接
二级	覆盖2个核心的安全层面：安全区域边界、安全计算环境。覆盖6个控制点的要求：接入控制、入侵防范、感知节点设备安全、网关节点设备安全、抗数据重放、数据融合处理。	1，感知节点安全防护2，网关节点安全防护
三级	覆盖2个核心的安全层面：安全区域边界、安全计算环境。覆盖6个控制点的要求：接入控制、入侵防范、感知节点设备安全、网关节点设备安全、抗数据重放、数据融合处理。	1，感知节点安全防护2，网关节点安全防护3，可信计算环境
三级（增强）	覆盖2个核心的安全层面：安全区域边界、安全计算环境。覆盖6个控制点的要求：接入控制、入侵防范、感知节点设备安全、网关节点设备安全、抗数据重放、数据融合处理。	1，感知节点安全防护2，网关节点安全防护3，可信计算环境
等保2.0咨询	等保2.0的咨询服务。	在线申请
安全自检	固件安全检测	固件安全检测

2，如何申请试用

试用规格说明：

覆盖控制点	阿里云产品名称	产品介绍	控制台	试用规格	试用授权数量	有效期
接入控制、感知节点设备安全、抗数据重放	IoT设备身份认证（ID²）	*https://www.aliyun.com/product/iotid*	*https://iotid.console.aliyun.com/*	基础版	3	1年
入侵防范、网关节点设备安全、数据融合处理	IoT安全运营中心（SOC）	*https://iot.aliyun.com/products/linksoc*	*https://isoc.console.aliyun.com/*	SOC-DPS	不限制	1年

覆盖控制点	阿里云产品名称	产品介绍	控制台	试用规格	试用授权数量	有效期
感知节点设备安全	IoT可信执行环境（TEE）	*https://iot.aliyun.com/products/tee*	无	TEE-Pro	不限制	1年
安全自检（非等保2.0要求）	IoT固件安全检测（FSS）	*https://iot.aliyun.com/products/iotfss*	*https://fss.iot.aliyun.com/*	——	1	——

1，等保2.0-试用申请：*https://page.aliyun.com/form/act227790978/index.htm*

2，使用申请试用的阿里云账号，创建一个新产品：等保2.0试用。

- 登录*ID²控制台*
- 创建产品：使用管理—>产品管理—>创建产品，产品名称：等保试用。

3，查看试用申请结果。

- 登录*ID²控制台*查看授权：使用管理—>产品管理，在”等保试用“产品操作中单击”查看“，”授权信息“中未激活的数量为3。
- 登录*FSS控制台*
- IoT安全运营中心SOC、IoT可信执行环境TEE授权不显示，但是这并不影响您的试用。

4，按照第三章”3，如何对接“进行感知节点、网关节点的安全能力对接。

3，如何对接

感知节点安全防护、网关节点安全防护的对接文档：

- IoT设备身份认证（ID²）对接说明：*https://help.aliyun.com/document_detail/142734.html*
- IoT安全运营中心（SOC）对接说明：*https://help.aliyun.com/document_detail/134362.html*
- IoT可信执行环境（TEE）对接说明：*https://help.aliyun.com/document_detail/126337.html*
- IoT固件安全检测（FSS）使用说明：*https://help.aliyun.com/document_detail/128442.html*