IoT设备身份认证

API参考

API参考

设备端API手册

本文档描述ID²接口在设备端如何使用，ID² Client SDK封装了底层对ID²载体、接口的操作细节，应用只需调用该接口进行相关操作，即可访问设备端ID²服务。

一、ID²初始化

函数原型

```
int id2_client_init(void);
```

功能描述

ID² Client SDK初始化，使用ID²设备端其它API之前，首先需调用该API进行初始化操作。

参数描述

无

返回值

0: 成功

其他参见设备端错误码

二、获取ID²

函数原型

```
int id2_client_get_id(uint8_t* id, uint32_t* len);
```

功能描述

获取ID²字串

参数描述

名称	输入/输出	描述
id	out	存放ID²字串的起始地址，长度不小于24字节
len	in/out	入参为参数id²的buf长度，出参为ID²字串的实际长度

返回值

0: 成功

其他参见设备端错误码

三、获取设备认证码—挑战应答模式

函数原型

```
int id2_client_get_challenge_auth_code(const char* challenge, const uint8_t* extra, uint32_t extra_len, uint8_t* auth_code, uint32_t* auth_code_len);
```

功能描述

基于挑战应答模式生成设备端认证码，可以有选择性的携带额外数据（extra），携带的extra参与设备认证码的签名运算。

参数描述

名称	输入/输出	描述
challenge	输入	挑战字起始地址，经SP Server从ID² Server获取
extra	输入	额外数据起始地址，可选，由Device和SP Server根据某种约定分别生成
extra_len	输入	额外数据长度，最大长度512字节
auth_code	输出	设备认证码起始地址，长度不小于256字节

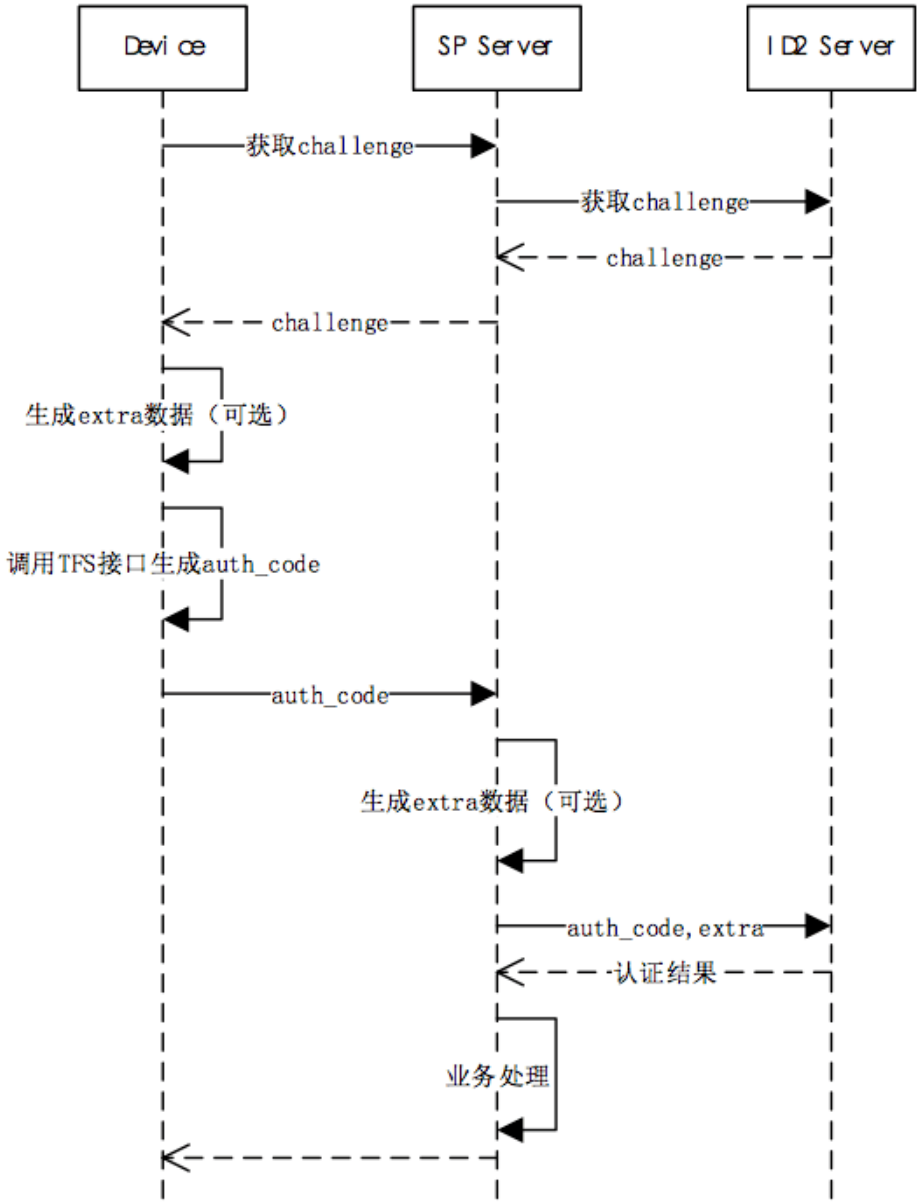
auth_code_len	输入/输出	入参为参数auth_code的buf长度，出参为设备认证码的实际长度

返回值

0: 成功

参见设备端错误码

交互流程图



四、获取设备认证码—时间戳模式

函数原型

```
int id2_client_get_timestamp_auth_code(const char* timestamp, const uint8_t* extra, uint32_t extra_len, uint8_t* auth_code, uint32_t* auth_code_len);
```

功能描述

基于时间戳模式生成设备端认证码，可以有选择性的携带额外数据（extra），携带的extra参与设备认证码的签名运算。时间戳可以由设备端RTC生成，或者由SP携带发送给设备端。

参数描述

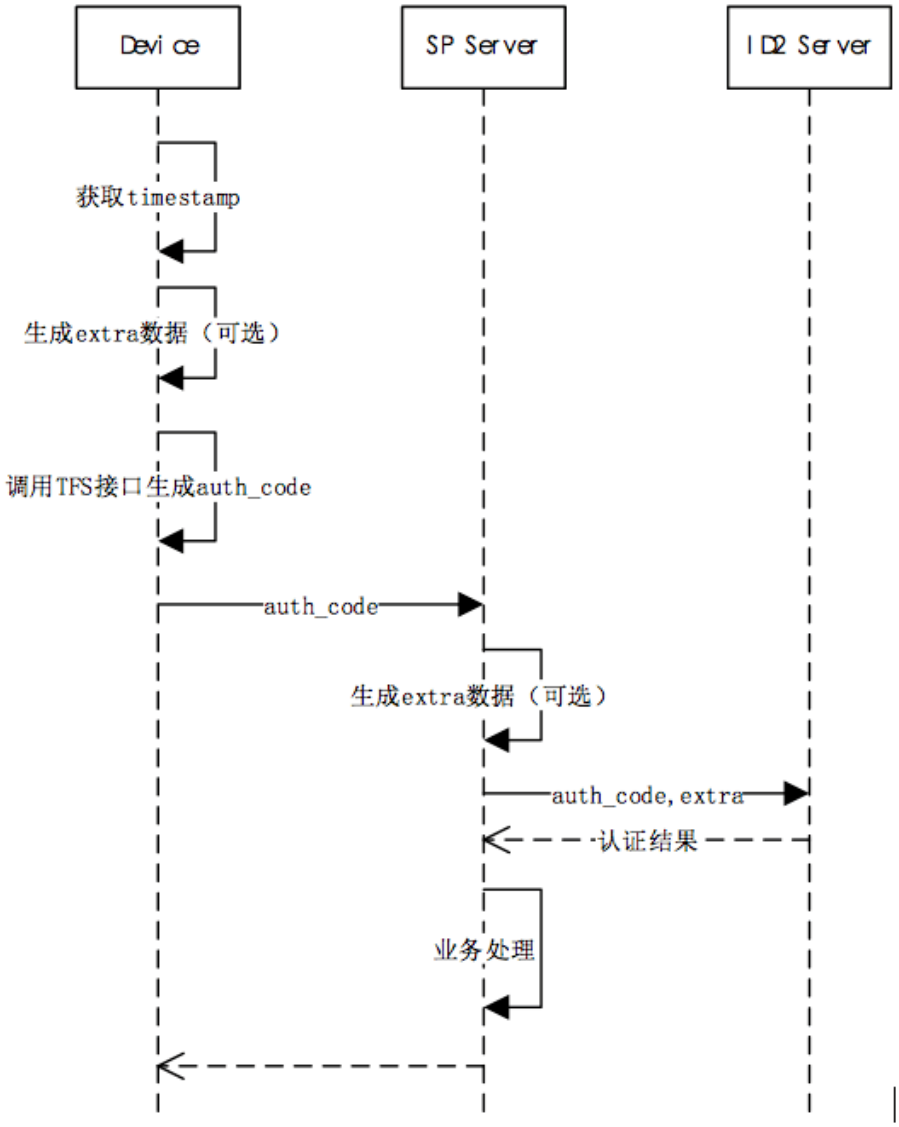
timestamp输入当前系统时间（从1970年1月1日午夜开始经过的毫秒数），字符串形式表示，如”1500954672653”extra输入额外数据起始地址，可选，由Device和SP Server根据某种约定分别生成extra_len输入额外数据长度，最大长度512字节auth_code输出设备认证码起始地址，长度不小于256字节len输入/输出入参为参数auth_code的buf长度，出参为设备认证码的实际长度

返回值

0: 成功

参见设备端错误码

交互流程图



五、ID²解密

函数原型

```
int id2_client_decrypt(const uint8_t* in, uint32_t in_len, uint8_t* out, uint32_t* out_len);
```

功能描述

使用ID²解密指定的数据

参数描述

名称	输入/输出	描述
in	输入	待解密的数据起始地址，数据如

		果是base64编码，需要先进行base64解码
in_len	输入	待解密的数据长度，长度不超过512字节
out	输出	解密后的数据起始地址
out_len	输入/输出	入参为参数out的buf长度，出参为解密后的数据长度

返回值

0: 成功

参见设备端错误码

六、获取设备烧录状态

函数原型

```
irot_result_t id2_client_get_prov_stat(bool *is_prov);
```

功能描述

获取设备端ID²的烧录状态

参数描述

名称	输入/输出	描述
is_prov	输出	布尔类型，存放ID²的烧录状态

返回值

0: 成功

参见设备端错误码

七、获取设备空发认证码

函数原型

```
irot_result_t id2_client_get_otp_auth_code(const uint8_t *token, uint32_t token_len, uint8_t *auth_code, uint32_t
```

```
*len);
```

功能描述

获取设备端ID²空发的认证码

参数描述

名称	输入/输出	描述
token	输入	ID² Server颁发的空发token, 按产品划分
token_len	输入	ID²空发token的长度，固定值32字节
auth_code	输出	ID²空发认证码的内存，长度小于256字节
len	输入/输出	入参为auth_code的内存大小，出参为认证码的实际长度

返回值

0: 成功

参见设备端错误码

八、烧录空发数据

函数原型

```
irot_result_t id2_client_load_otp_data(const uint8_t *otp_data, uint32_t len);
```

功能描述

烧录空发数据到设备中

参数描述

名称	输入/输出	描述
otp_data	输入	ID² Server下发的空发数据包（加密保护）
len	输入	ID²空发数据包的长度

返回值

0: 成功

参见设备端错误码

服务端API手册

一、服务端认证接口

接口名称

verify

访问地址

需要POP SDK支持，endpoint domain: id2.cn-shanghai.aliyuncs.com

功能描述

ID² 运行时认证接口

请求参数

名称	类型	是否必须	描述
apiVersioin	String	是	api版本号，当前取值1.1.2
id2	String	是	ID²标识
authCode	String	是	设备端生成的认证码，具体获取方式请参考设备端API对接指南
extra	String	否	与认证码关联的辅助认证数据
productKey	String	是	产品标识，从ID²控制台的产品列表中获取

返回值

```
成功：
{
"code":200,
"requestId":"F6AFB45A-0FD1-405E-AD2A-C50E34C429E5",
"success": true,
"data": true
}
失败：
{
"code":33,
"requestId":"F6AFB45A-0FD1-405E-AD2A-C50E34C429E5",
"success": false
}
```

错误码

参见服务端错误码

示例

示例Java code如下，相关的库代码请联系合作项目的阿里云IoT接口人

```
package com.aliyun.id2.demo;

import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.http.X509TrustAll;
import com.aliyuncs.id2.model.v20170707.VerifyRequest;
import com.aliyuncs.id2.model.v20170707.VerifyResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;

public class VerifyDemo {

private static String ACCESS_KEY = ""; // your access key
private static String ACCESS_SECRET = ""; // your access secret

public static void main(String[] args) throws ClientException {
try {
IClientProfile profile = DefaultProfile.getProfile("cn-shanghai", ACCESS_KEY, ACCESS_SECRET);
DefaultProfile.addEndpoint("cn-shanghai", "cn-shanghai", "ID2", "id2.cn-shanghai.aliyuncs.com");

IAcsClient client = new DefaultAcsClient(profile);
X509TrustAll.ignoreSSLCertificate();

String id2 = "00AAABBB11122281FE15B400";
String authCode =
"3~0~1245DC626946A9E5~1499753003564~Pfyt01tFpOFfI9HkHYjvqikoZpdch44U22ckmpvuwL1QfppOOIIGboFmfy
hnrX73hGvQ5BKzX1Acie+8MHoKZ64Y8tWMFMtbm3tmEqUzfSnYvGAEu/+YcytuZTKydh9ijJLUVeRgKUeS29q1zj9LOT
yGBXOxdesb9n9oQ225+3M=";
String extra = "digest1234";
```

```
VerifyRequest req = new VerifyRequest();
req.setId2(id2);
req.setAuthCode(authCode);
req.setExtra(extra);
req.setApiVersion("1.1.2");
req.setProductKey("xxxxxxxxxxx");

VerifyResponse response = client.getAcsResponse(req);
System.out.println("requestId:" + response.getRequestId());
if (response.getSuccess()) {
System.out.println("success, data:" + response.getData());
} else {
System.out.println("not success, code:" + response.getCode());
}
} catch (ServerException e) {
e.printStackTrace();
} catch (ClientException e) {
e.printStackTrace();
}
}
}
```

二、服务端认证并加密接口

接口名称

verifyAndEncrypt

访问地址

需要POP SDK支持，endpoint domain: id2.cn-shanghai.aliyuncs.com

功能描述

ID² 运行时认证并加密接口

请求参数

名称	类型	是否必须	描述
apiVersioin	String	是	api版本号，当前取值1.1.2
id2	String	是	ID²标识
authCode	String	是	设备端生成的认证码，具体获取方式请参考设备端接口描述
extra	String	否	与认证码关联的辅助认证数据

data	String	是	待加密的数据
productKey	String	是	产品标识，从ID²控制台的产品列表中获取

返回值

```
成功：
{
"code":200,
"requestId":"F6AFB45A-0FD1-405E-AD2A-C50E34C429E5",
"success": true,
"data": "MIGfM****DAQAB"
}
失败：
{
"code":33,
"requestId":"F6AFB45A-0FD1-405E-AD2A-C50E34C429E5",
"success": false
}
```

错误码

参见服务端错误码

示例

示例Java code如下，相关的库代码请联系合作项目的阿里云IoT接口人

```
package com.aliyun.id2.demo;

import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.http.X509TrustAll;
import com.aliyuncs.id2.model.v20170707.VerifyAndEncryptRequest;
import com.aliyuncs.id2.model.v20170707.VerifyAndEncryptResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;

public class VerifyAndEncryptDemo {
private static String ACCESS_KEY = ""; // your access key
private static String ACCESS_SECRET = ""; // your access secret

public static void main(String[] args) throws ClientException {
try {
IClientProfile profile = DefaultProfile.getProfile("cn-shanghai", ACCESS_KEY, ACCESS_SECRET);
DefaultProfile.addEndpoint("cn-shanghai", "cn-shanghai", "ID2", "id2.cn-shanghai.aliyuncs.com");
```

```
IAcsClient client = new DefaultAcsClient(profile);
X509TrustAll.ignoreSSLCertificate();

String id2 = "00AAABBB11122281FE15B400";
String authCode =
"3~0~1245DC626946A9E5~1499753003564~Pfyt01tFpOFfI9HkHYjvqikoZpdch44U22ckmpvuwL1QfppOOIIGboFmfy
hnrX73hGvQ5BKzX1Acie+8MHoKZ64Y8tWMFMtbm3tmEqUzfSnYvGAEu/+YcytuZTKydh9ijJLUVeRgKUeS29q1zj9LOT
yGBXOxdesb9n9oQ225+3M=";
String extra = "digest1234";

VerifyAndEncryptRequest req = new VerifyAndEncryptRequest();
req.setId2(id2);
req.setAuthCode(authCode);
req.setExtra(extra);
req.setApiVersion("1.1.2");
req.setData("1234");
req.setProductKey("xxxxxxxxxxx");

VerifyAndEncryptResponse response = client.getAcsResponse(req);
System.out.println("requestId:" + response.getRequestId());
if (response.getSuccess()) {
System.out.println("success, data:" + response.getData());
} else {
System.out.println("not success, code:" + response.getCode());
}
} catch (ServerException e) {
e.printStackTrace();
} catch (ClientException e) {
e.printStackTrace();
}
}
}
```

三、获取服务端随机数接口

接口名称

getServerRandom

访问地址

需要POP SDK支持，endpoint domain: id2.cn-shanghai.aliyuncs.com

功能描述

获取服务端随机数/挑战字

请求参数

名称	类型	是否必须	描述

apiVersioin	String	是	api版本号，当前取值1.1.2
id2	String	是	ID²标识

返回值

```
成功：
{
"code" : 200,
"requestId":"F6AFB45A-0FD1-405E-AD2A-C50E34C429E5",
"success": true,
"data": "6F5DDB5F21C28F06484A4695FAB915AA"
}
失败：
{
"code":18,
"requestId":"F6AFB45A-0FD1-405E-AD2A-C50E34C429E5",
"success": false
}
```

错误码

参见服务端错误码

示例

示例Java code如下，相关的库代码请联系合作项目的阿里云IoT接口人

```
package com.aliyun.id2.demo;

import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.exceptions.ServerException;
import com.aliyuncs.http.X509TrustAll;
import com.aliyuncs.id2.model.v20170707.GetServerRandomRequest;
import com.aliyuncs.id2.model.v20170707.GetServerRandomResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;


public class GetServerRandomDemo {

private static String ACCESS_KEY = ""; // your access key
private static String ACCESS_SECRET = ""; // your access secret

public static void main(String[] args) throws ClientException {
try {
IClientProfile profile = DefaultProfile.getProfile("cn-shanghai", ACCESS_KEY, ACCESS_SECRET);
```

```
DefaultProfile.addEndpoint("cn-shanghai", "cn-shanghai", "ID2", "id2.cn-shanghai.aliyuncs.com");

IAcsClient client = new DefaultAcsClient(profile);
X509TrustAll.ignoreSSLCertificate();

String id2 = "00AAABBB11122281FE15B400";

GetServerRandomRequest req = new GetServerRandomRequest();
req.setId2(id2);
req.setApiVersion("1.1.2");


GetServerRandomResponse response = client.getAcsResponse(req);
System.out.println("requestId:" + response.getRequestId());

if (response.getSuccess()) {
System.out.println("success, data:" + response.getData());
} else {
System.out.println("not success, code:" + response.getCode());
}
} catch (ServerException e) {
e.printStackTrace();
} catch (ClientException e) {
e.printStackTrace();
}
}
}
```

四、获取服务端认证码接口

接口名称

getServerAuthCodeAndEncryptData

访问地址

需要POP SDK支持，endpoint domain: id2.cn-shanghai.aliyuncs.com

功能描述

验证设备认证码有效性，获取服务端认证码，并加密数据

请求参数

名称	类型	是否必须	描述
id2	String	是	ID²标识
deviceAuthCode	String	是	设备端生成的认证码，服务端通过其验证设备合法性

deviceExtra	String	否	与设备认证码关联的辅助认证数据
data	String	是	待加密的数据
deviceChallenge	String	是	设备挑战字，用于生成服务端认证码
serverExtra	String	否	与服务端认证码关联的辅助认证数据
apiVersion	String	是	api版本号，当前取值1.1.2
productKey	String	是	产品标识，从ID²控制台的产品列表中获取

返回值

```
成功：
{
"code": 200,
"data": {
"serverAuthCode": "10~2~3DDFA7A45590CF12~QaB/DeZhx4KpBahW***gAZ5Q==",
"encryptData": "3s+wT***x4="
},
"success": true,
"requestId":"F6AFB45A-0FD1-405E-AD2A-C50E34C429E5"
}
失败：
{
"code": 34,
"data": null,
"success": false,
"requestId":"F6AFB45A-0FD1-405E-AD2A-C50E34C429E5"
}
```

错误码

参见服务端错误码

示例

示例Java code如下，相关的库代码请联系合作项目的阿里云IoT接口人

```
try {
String dataToEncrypt = "this is a String to encrypt!";
String serverExtra = "1234abcfe";

GetServerAuthCodeAndEncryptDataRequest req = new GetServerAuthCodeAndEncryptDataRequest();
req.setId2(id2);
```

```
req.setDeviceAuthCode(authCode);
if (extra != null) {
req.setDeviceExtra(extra); //no extra
}
req.setApiVersion("1.1.2");
req.setData(dataToEncrypt);
req.setDeviceChallenge("ABCD12348");
req.setServerExtra(serverExtra);
req.setProductKey("xxxxxxxxxx");

GetServerAuthCodeAndEncryptDataResponse response = client.getAcsResponse(req);
if (response.getSuccess() && response.getCode() == 200) {
byte[] decryptedArray =
id2utils.getKeyService(cryptType).decrypt(Base64.decodeBase64(id2s[index].getPrivateKey()),
Base64.decodeBase64(response.getData().getEncryptData()));
String decryptedString = new String(decryptedArray);
//using id2utils.getKeyService(cryptType).decrypt to decrypt and verify encrypted data
//using tfsVerifyServer to verify server authcode
return dataToEncrypt.equals(decryptedString) && tfsVerifyServer(index, response.getData().getServerAuthCode(),
serverExtra);
}
} catch (ServerException e) {
e.printStackTrace();
} catch (ClientException e) {
e.printStackTrace();
}
```

五、ID²空发接口

接口名称

otpGetId2

访问地址

需要POP SDK支持，endpoint domain: id2.cn-shanghai.aliyuncs.com

功能描述

ID²及其密钥的空发

请求参数

名称	类型	是否必须	描述
apiVersioin	String	是	api版本号，当前取值1.1.2
deviceAuthCode	String	是	设备认证码，基于空发token和设备端SDK生成，具体生成方式请参

			考设备端接口描述

返回值

```
成功：
{
"code": 200,
"data": {
"provisionData": "ATAww……nNfro=", //空发数据，设备从中解析出ID²和密钥
"remaining": 100 //空发授权余量
},
"success": true,
"requestId": "F6AFB45A-0FD1-405E-AD2A-C50E34C429E5"
}
失败：
{
"code": 34,
"data": null,
"requestId": "F6AFB45A-0FD1-405E-AD2A-C50E34C429E5",
"success": false
}
```

错误码

参见服务端错误码

示例

示例Java code如下，相关的库代码请联系合作项目的阿里云IoT接口人

```
try {
OtpGetId2Request req = new OtpGetId2Request();
req.setApiVersion("1.1.2");
req.setDeviceAuthCode(deviceAuthCode);
OtpGetId2Response response = client.getAcsResponse(req);
Integer code = response.getCode();
if (response.getSuccess()) {
OtpGetId2Response.Data returnData = response.getData();
System.out.println("success, provData:" + returnData.getProvisionData() + "remaining:" +
returnData.getRemaining());
} else {
System.out.println("fail, code:" + response.getCode());
}
} catch (ServerException e) {
e.printStackTrace();
} catch (ClientException e) {
e.printStackTrace();
}
```

IROT硬件抽象层接口

硬件抽象层接口描述

硬件初始化

函数原型： irot_result_t irot_hal_init(void);

功能描述： SE芯片初始化（例如打开SE芯片供电，初始化SE芯片会话等）

参数描述： void

返回值： 见irot_result_t

读取 ID2的 ID

函数原型：irot_result_t irot_hal_get_id2(uint8_t *id2, uint32_t* len);

功能描述：读取 ID2的 ID

参数描述：[OUT] id2，ID 的起始地址 [IN/OUT] len，输入为缓冲区长度，输出为 ID 的实际长度。（HEX 数据格式， 当前长度为 12 字节）

返 回 值：见 irot_result_t

对称算法

函数原型：irot_result_t irot_hal_sym_crypto(key_object *key_obj, uint8_t key_id,*

const uint8_t iv, uint32_t iv_len,

const uint8_t *in, uint32_t in_len,*

uint8_t out, uint32_t *out_len,*

sym_crypto_param_t crypto_param);

功能描述：用对称算法实现数据的加解密

参数描述：[IN] key_obj， 密钥对象，见 key_object 类型

[IN] key_id， 密钥标识，索引内部密钥

[IN] iv，初始化向量地址

[IN] iv_len， 初始化向量长度

[IN] in，输入数据起始地址

[IN] in_len，输入数据长度

[OUT] out,，输出数据地址

[IN/OUT] out_len，输入为缓冲区大小，输出为真实数据长度

[IN] crypto_param，加解密参数结构，见 sym_crypto_param_t

备注：key_obj 和 key_id 两个参数为二选一使用，当 key_obj 参数不为 NULL，则使用此 参数作为密钥进行运算。当 key_obj 参数为 NULL，则使用key_id 标识内部密钥进行运算。 其它类似 API 同上。

非对称算法

私钥签名

函数原型：irot_result_t irot_hal_asym_priv_sign(key_object *key_obj, uint8_t key_id,*

const uint8_t in, uint32_t in_len,

uint8_t *out, uint32_t* out_len,

asym_sign_verify_t type);

功能描述：用私钥对数据进行签名

参数描述：[IN] key_obj， 密钥对象，见 key_object 类型

[IN] key_id， 密钥标识，索引内部密钥

[IN] in，待签名数据的起始地址

[IN] in_len，待签名数据的长度

[OUT] sign，签名数据的起始地址

[IN/OUT] sign_len，输入为缓冲区长度，输出为签名数据的实际长度

[IN] type，签名模式，见asym_sign_verify_t

返回值： 见irot_result_t

私钥解密

函数原型：irot_result_t irot_hal_asym_priv_decrypt(key_object *key_obj, uint8_t key_id,*

const uint8_t in, uint32_t in_len,

uint8_t *out, uint32_t* out_len,

asym_padding_t padding);

功能描述：用私钥对数据进行解密

参数描述：[IN] key_obj， 密钥对象，见 key_object 类型

[IN] key_id， 密钥标识，索引内部密钥

[IN] in，密文数据的起始地址

[IN] in_len，密文数据的长度

[OUT] out，明文数据的起始地址

[IN/OUT] out_len，输入为缓冲区长度，输出为明文数据的实际长度

[IN] padding，填充方式，见 asym_padding_t

返 回 值：见 irot_result_t

散列算法

函数原型：irot_result_t irot_hal_hash_sum(const uint8_t *in, uint32_t in_len,*

uint8_t out, uint32_t* out_len, digest_t type);

功能描述：用指定算法对数据做摘要

参数描述： [IN] in，原始数据的起始地址

[IN] in_len，原始数据的长度

[OUT] out，摘要数据的起始地址

[IN/OUT] out_len，输入为缓冲区长度，输出为摘要数据的实际长度

[IN] type，摘要算法，见 hash_t

返 回 值：见 irot_result_t

对称算法-块模式

```
typedef enum
{
    BLOCK_MODE_ECB = 0x00,
    BLOCK_MODE_CBC = 0x01,
    BLOCK_MODE_CTR = 0x02,
} block_mode_t;
```

对称算法-填充类型

```
typedef enum
{
    SYM_PADDING_NOPADDING = 0x00,
    SYM_PADDING_PKCS5 = 0x02,
    SYM_PADDING_PKCS7 = 0x03,
} irot_sym_padding_t;
```

非对称算法-填充类型

```
typedef enum
{
    ASYM_PADDING_NOPADDING = 0x00,
    ASYM_PADDING_PKCS1 = 0x01,
} irot_asym_padding_t;
```

加密解密类型

```
typedef enum
{
    MODE_DECRYPT = 0x00,
    MODE_ENCRYPT = 0x01,
} crypto_mode_t;
```

非对称算法-签名类型

```
typedef enum
{
    ASYM_TYPE_RSA_MD5_PKCS1 = 0x00,
    ASYM_TYPE_RSA_SHA1_PKCS1 = 0x01,
    ASYM_TYPE_RSA_SHA256_PKCS1 = 0x02,
    ASYM_TYPE_RSA_SHA384_PKCS1 = 0x03,
    ASYM_TYPE_RSA_SHA512_PKCS1 = 0x04,
```

```
    ASYM_TYPE_SM3_SM2 = 0x05,
    ASYM_TYPE_ECDSA = 0x06,
} asym_sign_verify_t;
```

哈希算法类型

```
typedef enum
{
    HASH_TYPE_SHA1 = 0x00,
    HASH_TYPE_SHA224 = 0x01,
    HASH_TYPE_SHA256 = 0x02,
    HASH_TYPE_SHA384 = 0x03,
    HASH_TYPE_SHA512 = 0x04,
    HASH_TYPE_SM3 = 0x05,
} hash_t;
```

对称算法加解密参数

```
typedef struct _sym_crypto_param_t
{
    cipher_t cipher_type; ///< cipher type
    block_mode_t block_mode; ///< block mode
    sym_padding_t padding_type; ///< padding type
    mode_t mode; ///< mode(encrypt or decrypt)
} sym_crypto_param_t;
```

通用-密钥对象

```
typedef struct
{
    struct
    {
    uint8_t key_object_type; ///< the key object type
```

```
    } head;
    struct
    {
    uint8_t buf[0x04]; ///< placeholder for key
    } body;
} key_object;
```

对称算法-密钥对象

```
typedef struct
{
    struct
    {
        uint8_t key_object_type;
    } head;
    struct
    {
        uint8_t key_value; ///< the key value
        uint32_t key_len; ///< the key length(bytes)
    } body;
} key_object_sym;
```

RSA 公钥-密钥对象

```
typedef struct
{
    struct
    {
        uint8_t key_object_type;
    } head;
    struct
```

```
    {
        uint8_t e; ///< public exponent
        uint32_t e_len; ///< public exponent length(bytes)
        uint8_t n; ///< public modulus
        uint32_t n_len; ///< public modulus length(bytes)
    } body;
} key_object_rsa_public;
```

RSA 私钥-密钥对象

```
typedef struct
{
    struct
    {
        uint8_t key_object_type;
    } head;
    struct
    {
        uint8_t d; ///< private exponent
        uint32_t d_len; ///< private exponent length(bytes)
        uint8_t n; ///< private modulus
        uint32_t n_len; ///< private modulus length(bytes)
    } body;
} key_object_rsa_private;
```

RSA CRT 格式私钥-密钥对象

```
typedef struct
{
    struct
```

```
    {
        uint8_t key_object_type;
    } head;
    struct
    {
        uint8_t p; ///< 1st prime factor
        uint8_t q; ///< 2st prime factor
        uint8_t dp; ///< d % (p - 1)
        uint8_t dq; ///< d % (q - 1)
        uint8_t qinv; ///< (1/q) % p
        uint32_t len; ///< the length for the 5 parameters must with the same length(bytes)
    } body;
} key_object_rsa_crt_private;
```

附录

错误码定义

```
typedef enum
{
    IROT_SUCCESS = 0, ///< The operation was successful.
    IROT_ERROR_GENERIC = -1, ///< Non-specific cause.
    IROT_ERROR_BAD_PARAMETERS = -2, ///< Input parameters were invalid.
    IROT_ERROR_SHORT_BUFFER = -3, ///< The supplied buffer is too short for the output.
    IROT_ERROR_EXCESS_DATA = -4, ///< Too much data for the requested operation was passed.
    IROT_ERROR_OUT_OF_MEMORY = -5, ///< System out of memory resources.
    IROT_ERROR_COMMUNICATION = -7, ///< Communication error
    IROT_ERROR_NOT_SUPPORTED = -8, ///< The request operation is valid but is not supported in this implementation.
    IROT_ERROR_NOT_IMPLEMENTED = -9, ///< The requested operation should exist but is not yet implementation.
```

```
    IROT_ERROR_TIMEOUT = -10,///< Communication Timeout
    IROT_ERROR_ITEM_NOT_FOUND = -11,///< Id2 is not exist
} irot_result_t;
```

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

安全SDK适配说明

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<br />
```

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)

- 返回值：

成功：= 0
失败：-1

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。
- 参数：

名称	数据类型	描述
flag	int	分区的读写权限

- 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
EXTRA_TARGET_MAKEFILES +=  $($(HOST_MCU_FAMILY)_LOCATION)/download.mk
```

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。

SE 芯片驱动API文档

概述

本文档主要提供给SE安全芯片厂商，厂商需要在SE驱动接口里面实现和SE芯片基础通讯和数据交互。

SE 驱动接口

se_open_session

函数原型： irot_result_t se_open_session(void ** handle)
功能描述： 与SE芯片建立会话连接
参数描述： [IN]handle，会话句柄
返回值： 见附录irot_result_t

se_transmit

函数原型： irot_result_t se_transmit(void *handle,*
const uint8_t cmd_apdu,
const uint32_t cmd_len,
uint8_t *rsp_buf,*
uint32_t rsp_len);
功能描述： 发送 APDU 命令到 SE 芯片，并接收 APDU 响应，响应数据包含 2 字节状态字。该接口阻塞等待接收 响应数据，或接收到错误的返回值。
参数描述： [IN] handle，会话句柄
[IN] cmd_apdu, 命令APDU，参考规范ISO7816-4
[IN] cmd_len, 命令APDU长度
[IN] rsp_buf, APDU返回数据Buffer
[INOUT] rsp_len, 输入APDU Buffer的长度，输出为实际的APDU数据返回长度（包含2个字节状态字）
返回值： 见附录irot_result_t

se_close_session

函数原型： irot_result_t se_close_session(void** handle);

功能描述： 关闭会话连接
参数描述: [IN] handle, 会话句柄
返回值： 见附录irot_result_t

备注： 接口中的 handle 参数，在通讯交互过程中，可用于保存会话的状态信息。

错误码定义

错误码	值	含义
IROT_SUCCESS	0	调用成功
IROT_ERROR_GENERIC	-1	通用错误
IROT_ERROR_BAD_PARAMETERS	-2	输入参数错误
IROT_ERROR_SHORT_BUFFER	-3	数据缓冲长度不够
IROT_ERROR_EXCESS_DATA	-4	数据长度超出请求的操作
IROT_ERROR_OUT_OF_MEMORY	-5	无法分配内存
IROT_ERROR_COMMUNICATION	-7	通讯错误
IROT_ERROR_NOT_SUPPORTED	-8	不支持的操作
IROT_ERROR_NOT_IMPLEMENTED	-9	对应的操作没有实现
IROT_ERROR_TIMEOUT	-10	通信超时
IROT_ERROR_ITEM_NOT_FOUND	-11	没有找到ID2