

物联网设备身份认证

开发指南

产品文档



腾讯云

【 版权声明 】

©2013–2023 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100。

文档目录

开发指南

认证与通信流程

设备端接入开发

服务端接入开发

开发指南

认证与通信流程

最近更新时间：2020-12-14 18:26:03

本文主要介绍认证与通信过程中涉及到的各角色信息以及逻辑流程。

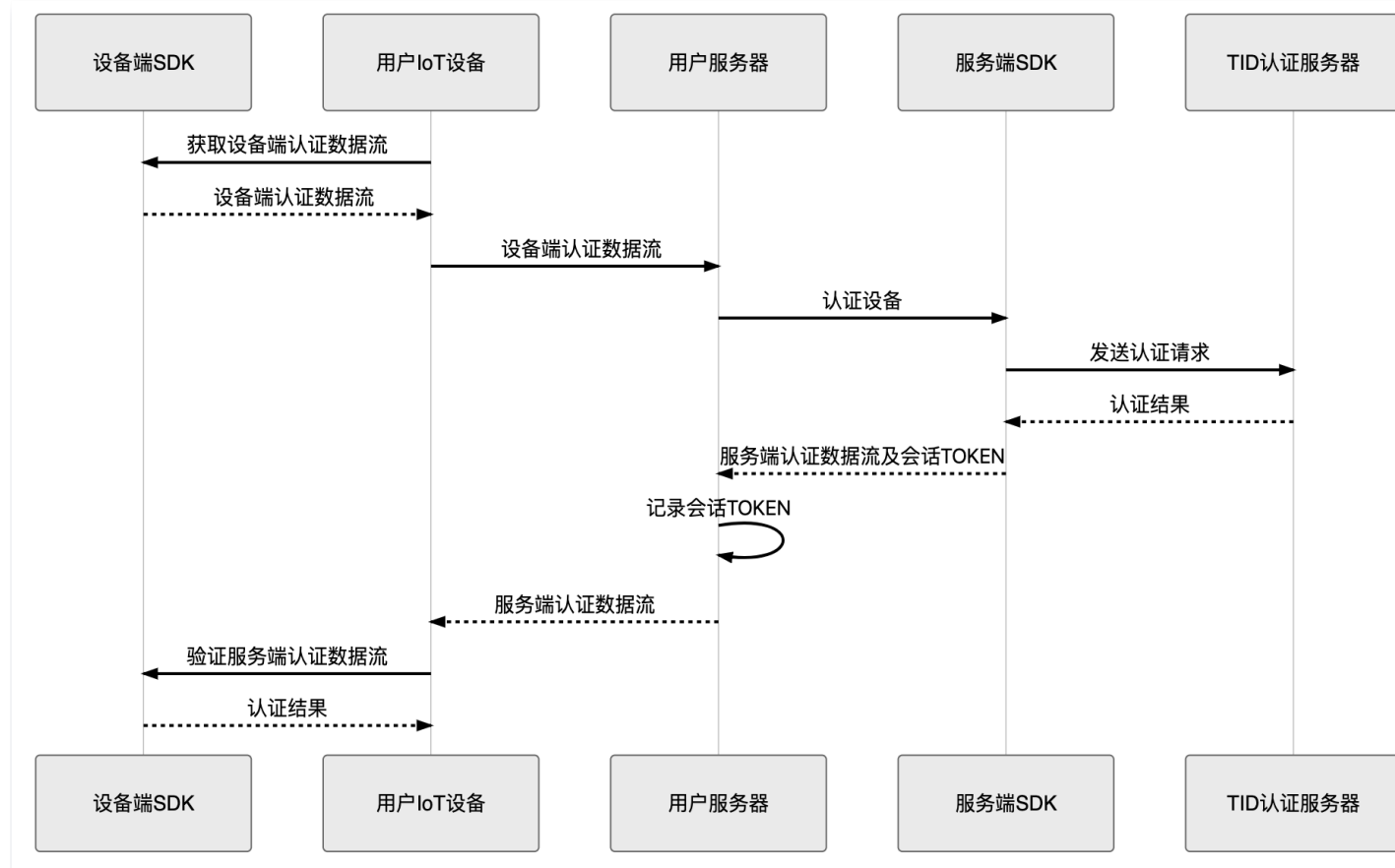
角色介绍

本节主要介绍 IoT TID 认证与通信流程中涉及的角色信息。

- 用户 IoT 设备：设备商生产或运营的物联网终端设备。
- 设备端 SDK：对接用户 IoT 设备的 C 语言 SDK，用于协议的打包与解包。
- 用户服务端：设备商自己部署的服务器，负责处理业务逻辑，连接用户 IoT 设备与 TID 认证服务器。
- 服务端 SDK：对接用户服务端的多语言 SDK，封装了 TID 认证 API 及加解密 API。
- TID 认证服务器：腾讯云 IoT TID 认证服务平台，负责处理 TID 认证请求。

IoT TID 认证流程

本节主要介绍 IoT TID 认证过程中的逻辑流程。

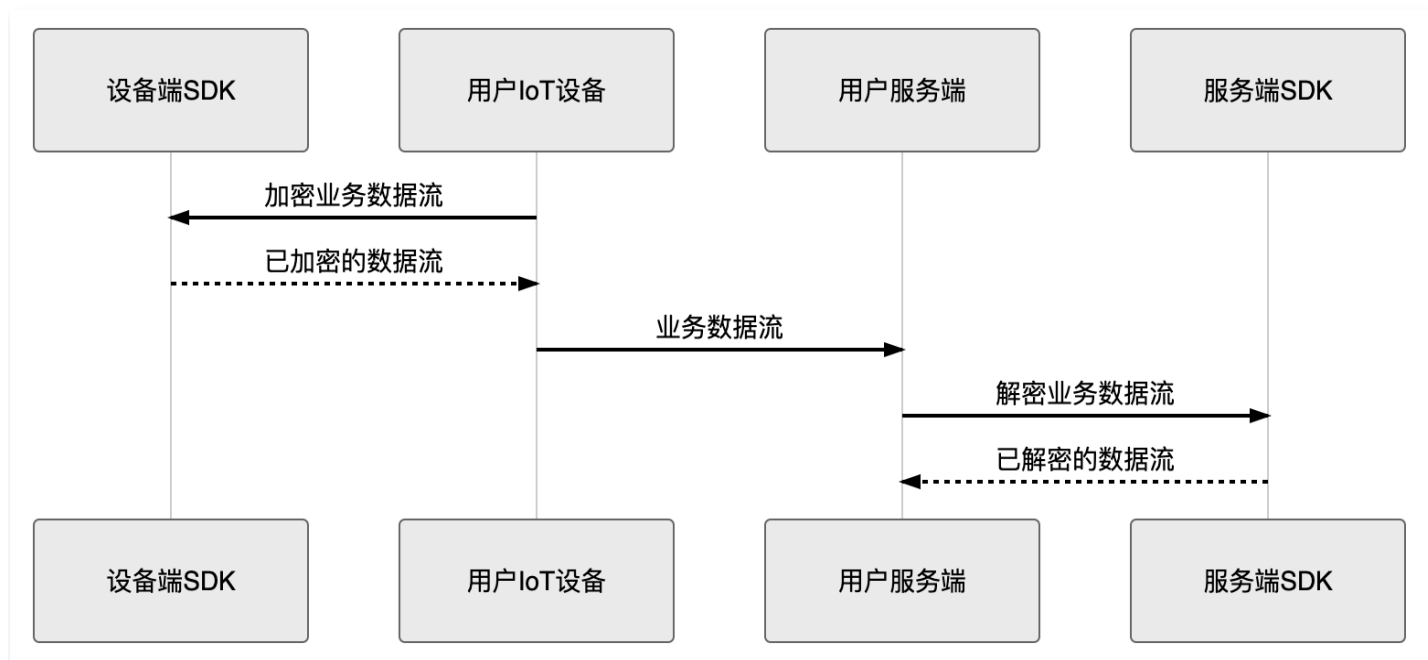


1. 设备端通过 SDK 获取认证数据流。
2. 设备端将认证数据流传输至用户服务器。
3. 用户服务器通过服务端 SDK 将认证请求，发送至 IoT TID 认证服务器。
4. TID 认证服务器返回认证结果，若认证成功，服务端 SDK 会返回服务端认证数据流和会话 Token。
5. 若服务端 SDK 返回成功，用户服务器需要将服务端认证数据流完整传输至设备端，同时记录会话 Token 用于后续的业务数据加解密。

IoT TID 通信流程

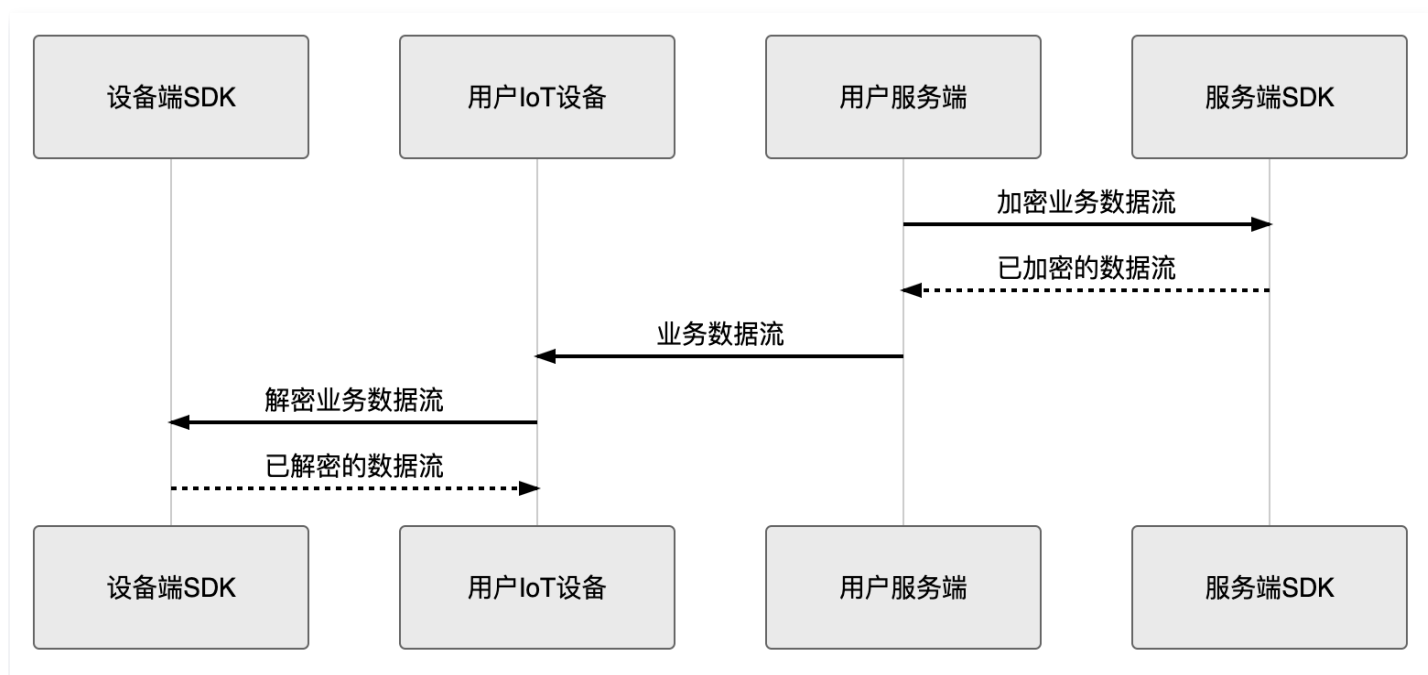
本节主要介绍 IoT TID 通信过程中的逻辑流程。

设备端数据上行



1. 设备端调用 SDK 的对称算法加密接口，传入原始业务数据流和会话 Token，接口会返回加密后的数据流。
2. 设备端将加密后的业务数据流完整传输至用户服务器。
3. 用户服务器调用服务端 SDK 的对称算法解密接口，传入加密后的数据流和会话 Token，接口会返回解密后的数据流。
4. 用户服务器根据解密后的业务数据流处理业务流程。

服务端数据下行



1. 用户服务器调用服务端 SDK 的对称算法加密接口，传入原始业务数据流和会话 Token，接口将返回加密后的数据流。
2. 用户服务器将加密后的业务数据流完整传输至设备端。
3. 设备端调用 SDK 的对称算法解密接口，传入加密后的数据流和会话 Token，接口将返回解密后的数据流。
4. 设备端根据解密后的业务数据流处理业务流程。

设备端接入开发

最近更新时间：2021-08-24 11:34:41

操作场景

腾讯云 TID 接入类型分为三种：SE（Secure Element）安全芯片、TEE（Trusted Execution Environment）可信执行环境、WBC（White-Box Cipher）软加固。

在使用之前，请您先根据设备使用场景选择合适的安全等级，从软加固 > TEE > SE，安全等级依次提高。

- 对于安全高度敏感的设备，建议您使用最高等级的 SE 方式，例如，智能门锁产品。
- 如果采用 SE 方式，您需要向腾讯云的 [合作伙伴](#) 采购安全芯片并集成至您的设备中。
- 如果采用 TEE 或软加固方式，则通过本控制台即可完成自助服务。

SE 接入方式

本文档介绍设备端如何使用 SE 方式接入。

下载 SDK

下载地址：[设备端 SDK](#)。

设备端 SDK 包括以下两部分：

- SE 芯片 SDK：提供调用 SE 芯片安全能力的 API，不同厂家/型号的 SE 芯片具备不同的 SDK。
- TID 安全协议：定制优化的轻量级 TLS 协议实现的 TID-TLS，基于 TID 实现了 TCP 链路层的安全。

版本说明

设备端 SDK 版本说明如下：

- 版本：V1.0 版本。
- 发布日期：2019/06/19。
- 开发环境：Linux GNU、Keil、IAR。
- 编程语言：C 语言。

主要接口

设备端使用 TID 过程中涉及到的主要函数接口说明如下：

TID 初始化

- 函数：`status_t tid_client_init()`
- 参数说明：无
- 返回：函数执行的状态，0为成功，其余请参考 [错误状态码](#) 说明。

获取设备认证码

- 函数: `status_t tid_client_get_authcode( uint8_t *authcode, size_t *buf_len)`
- 参数说明:
 - authcode: 获取从设备端生成的 authcode 的 buffer。
 - buf_len: authcode buffer 的长度, 同时用于返回实际生成的 authcode 的长度。
- 返回: 函数执行的状态, 0为成功, 其余请参考 [错误状态码](#) 说明。

认证设备状态码

- 函数:
`status_t tid_client_validate_authcode( const uint8_t *authcode, size_t in_buf_len, uint8_t *token, size_t *token_len)`
- 参数说明:
 - authcode: 输入的从 Server 端获取的待验证的 authcode。
 - in_buf_len: authcode 的长度。
 - token: 用于输出认证成功之后产生的 token。
 - token_len: 输出的 token 的长度。
- 返回: 函数执行的状态, 0为成功, 其余请参考 [错误状态码](#) 说明。

会话密钥导入

- 函数: `status_t tid_client_import_sesskey(uint8_t *sess_key, int key_len, int key_index)`
- 参数说明:
 - sess_key: 会话密钥。
 - key_len: session_key 长度。
 - key_index: 导入的 session index。
- 返回: 函数执行的状态, 0为成功, 其余请参考 [错误状态码](#) 说明。

会话数据加解密

- 函数:
`status_t tid_client_sym_crypto(const uint8_t *in_buf, uint8_t *out_buf, int in_buf_len, int key_index, int is_enc)`
- 入参:
 - in_buf: 输入数据。
 - out_buf: 加解密后的结果。
 - in_buf_len: 输入 buffer 的大小。
 - key_index: session 的 index。
 - is_enc: 加密还是解密, 1表示加密, 0表示解密。
- 返回: 函数执行的状态, 0为成功, 其余请参考 [错误状态码](#) 说明。

集成 SDK

按照以下步骤进行设备端 SDK 的集成：

1. 将 SDK 集成到设备的开发环境中，支持常见的嵌入式开发环境：嵌入式 Linux、Keil、IAR 等。
2. 支持 cmake 的环境（例如：嵌入式 Linux）可以通过 cmake 来生成 Makefile 进行编译，使用 IDE（例如：Keil 和 IAR）可使用 IDE 通过修改工程文件即可完成集成。

配置 SDK

根据您的应用场景修改 tid_config.h 中对应的配置项，主要的配置项包括：

- 是否开启 TID-TLS，TID-TLS 适用于支持通过 TCP/IP 接入网络的设备。
- 是否要求支持国密，默认为不支持，如需支持，需进行配置。
- 配置和安全芯片通信的接口，例如：I2C、SPI 等。

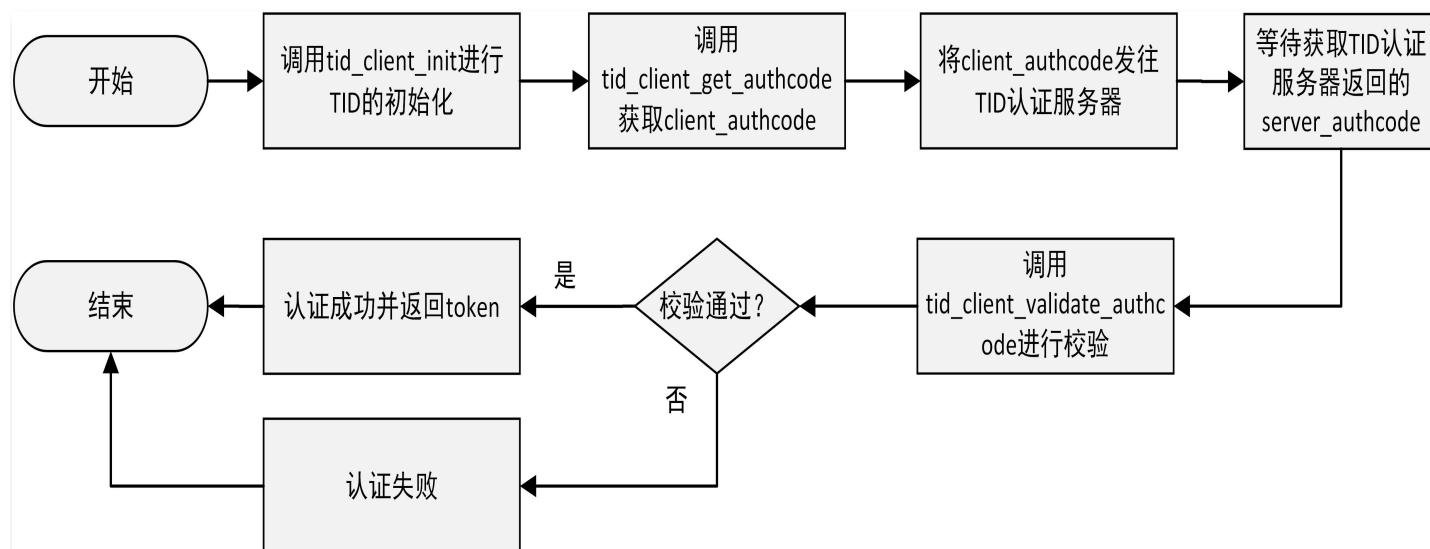
以下为配置使用 I2C 接口的复旦微安全芯片时，tid_config.h 的配置示例代码：

```
#define TID_SE_VENDOR TID_SE_VENDOR_FM
#define TID_BUS_TYPE TID_BUS_I2C
```



TID 双向校验

按照下列流程调用相关函数进行 TID 双向校验，函数的使用说明请参见本文 [设备端主要接口](#)。



说明：

服务端可不返回 Token，此时 SE 安全芯片将维持上一个 Token 不变，后续所有业务数据的加解密操作均由 SE 安全芯片完成。详情请参见 [设备端 SDK 相关代码](#)。

软加固接入方式

如果要使用软加固方式，需满足以下条件：

- 确保设备端上的操作系统支持动态链接。
- CPU 的芯片架构在控制台的主芯片架构列表中。

使用请求空发 TID 信息云 API 获取动态链接文件（.so）下载地址，下载到软加固的动态链接文件（.so）后，放置在 SDK 的 lib 目录下，在 tid_config.h 配置开启软加固的宏：

```
#define TID_SEC_TYPE    TID_SEC_SOFT
```



完成配置后，SDK 的使用流程和基于 SE 安全芯片方式接入 TID 的情况一致。

服务端接入开发

最近更新时间：2020-12-14 18:00:20

操作场景

本文档介绍服务端 SDK 的相关接口与功能。

操作步骤

下载 SDK

下载地址：[服务端 SDK](#)。

版本说明

Golang 语言

v20190724 版本

- 目录：Golang/v20190724
- 更新说明：lot Tid 设备认证支持 HTTP 和 HTTPS 两种协议的长连接模式；调整目录结构。
- 使用说明：
 - 设备认证：参考 src/iottid/server_test.go。
 - SM4 加解密：参考 src/iottid/sm4_test.go。

v20190501 版本

- 目录：Golang/v20190501
- 更新说明：初始版本，封装了 lot Tid 的设备认证接口和 SM4 国密算法加解密接口。
- 使用说明：
 - 设备认证：参考 src/iottid/server_test.go。
 - SM4 加解密：参考 src/iottid/sm4_test.go。

Python 语言

v20190724 版本

- 目录：Python3/v20190724
- 更新说明：lot Tid 设备认证支持 HTTP 和 HTTPS 两种协议的长连接模式；调整目录结构。
- 使用说明：
 - 设备认证：参考 iottid/server_test.py。
 - SM4 加解密：参考 iottid/sm4_test.py。

v20190501 版本

- 目录：Python3/v20190501
- 更新说明：初始版本，封装了 lot Tid 的设备认证接口和 SM4 国密算法加解密接口。
- 使用说明：
 - 设备认证：参考 iottid/server_test.py。
 - SM4 加解密：参考 iottid/sm4_test.py。

PHP 语言

v20190724 版本

- 目录：Php/v20190724
- 更新说明：lot Tid 设备认证支持 HTTP 和 HTTPS 两种协议的长连接模式；调整目录结构。
- 使用说明：
 - 设备认证：参考 tests/ServerTest.php。

v20190701 版本

- 目录：PHP/v20190701
- 更新说明：初始版本，封装了 lot Tid 的设备认证接口。
- 使用说明：
 - 设备认证，参考 tests/ServerTest.php。

开发流程

生成通信密钥

用户服务端与 TID 认证服务器之间的通信涉及到加密及签名验证，用户需要使用 [服务端 SDK](#) 提供的工具生成 RSA 公私钥，并将生成的公钥文件（public_key.pem）的完整内容，通过 [使用管理控制台](#) 上传至 TID 认证服务平台，同时保留私钥用于 SDK 调用。

生成通信密钥的工具详情如下所示：

- 目录：DevTools
- 使用方式：
 - 安装依赖库（requirements.txt）。
 - 执行 "python generate_rsa_key_pair.py KEY_PAIR_DIR"，KEY_PAIR_DIR 表示密钥存放的目录，该目录下会生成 public_key.pem 公钥文件和 private_key.pem 私钥文件。
- 公钥文件内容示例：

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxLCA8E4dWD9pgFBOCqW6
f2/TotqutryZJjgn8QsvXDMmZru05Wb3ts+qs/uM1BN2zAU27MZDsffsxXBn0VO0
awHYvLkeJcOTE0UrHc7tTWm7eQNG0hNrOYCnXmKXMRAGhhP2834OyqY2DM6UICAN
EF4bpSNhG7ctOCXhNqW1xDiRpJQ+5+1ob8L6801hMCEsQUADxu5nsohGlc5RGw0G
```



```
xh5mcb2mEdN66ha8bGk78ntwGZEUudKQZXDEW4+lf+kl3lfjAiUfPipBYeexKVH
fhYrWhqBwB5ahxRFCA+Zasowle71OgliRu9Rto2Yda/e538l0iRDllmkSDrzQSdB
xQIDAQAB
-----END PUBLIC KEY-----
```

- 私钥文件内容示例：

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAxLCA8E4dWD9pgFBOCqW6f2/TotqutryZJjgn8QsvXDMmZru0
5Wb3ts+qs/uM1BN2zAU27MZDsffsxXBn0VO0awHYvLkeJcOTE0UrHc7tTWm7eQNG
0hNrOYCnXmKXMRAGhhP2834OyqY2DM6UICANEF4bpSNhG7ctOCXhNqW1xDiRpJQ+
5+1ob8L6801hMCEsQUADxu5nsohGlc5RGw0Gxh5mcb2mEdN66ha8bGk78ntwGZE
uudKQZXDEW4+lf+kl3lfjAiUfPipBYeexKVHfhYrWhqBwB5ahxRFCA+Zasowle71
OgliRu9Rto2Yda/e538l0iRDllmkSDrzQSdBxQIDAQABAoIBAQCoio6SXH1sxznC
SA33tbevX8VUQYuA7u9U/r3LIUh9+x6m3aXPBYKy1hRykdVFTUB45ub/0YzkjYwl
S5/Kka2CHFz6qq4YshPvaU/fBRf3Q+jvc01WamlxJrtEd7sSwaa1cZolYBmQsQiT
xNjNBmnYpjR0xU9SKOo/1yTFUBCWkvWUxnty38FsnOfU83liq3/eSMGODc5ttc3
YWoe2Cj+BX2gbgLRSDtGuQfGTWklGxvQ58SRnUeR3o84pRJWixGcJc39sJlxej
oEa3GXkwjtqUCcxzW/TTMicroJcj6fvs/yM8BITwLvBv6LMTj9jPjbzvB+ug/N
nnT/Rr7pAoGBAME2jtFv0l4DmNlepAXzlrX5M+pmUG6lliNn7N3AHKRXXKV5MSeTq
PrHpL//RgETgjUoEk6R6lhuChXrflxCzmcoorxFishk8eq48FJ2ezrWAYj5Vrkhu
MQPQKLjjxZ8LGGSSe7UOAavvcSuXl1AheMVtXMhoXwAY0ww2lrxwvd5vrAoGBAPwf
yZA4NyLQhO6LQ6+pTa2ogFI0n6BDU+BOiFw5UFDT0VurMWvy9pfuvalqo7sAqKbD
EhRn3+LeOxN6pwiA89XKMILuoPEFPUCAmqvSDpMcXzkoiwmffms5C2RT9sQH4ycu
bQ8XzqTn9P3v9T4Hy6mKyQ63DtHo8qHxD9N4Ep0PAoGBAKrTC1w3krAZ0Y70dcYl
DuaqTu7LLzscm4eTaDbREW68o0Tmajpk5tWUyRq/H0FKnHqwaX1BfpUbUoMkZvnB
NeWGYSCelsAhTCpmd5fO866qhm9wiE+p+UDytzJWmX4NpoK4DRpz7rdx2Zqs6jhy
ITo05NEOyZBFZyXwN1UbgOqbAoGBAKjMC2HQETHw0x72BDIGMqhyaMP7baeZi/1f
9rFv+dKzjbZGzkbMmsYro7VaXRYIICtmUz1wMhes23Z2JoBt/VIEFI8pvny/Gjwm
Hee+38AsPaCCAVxG+ReGUJcyE8M42J7x2ThCoyoK1h5xWgrWo0b2URfV/UkSlefE
Ael9uVCHAOgAXy8Lm+OkJPgAqAcZ2HLoOTvnNITYVQ6wb1c5Inv4CNF8EW4eUG+7
5nTXTDcB70iloCwHc2b9UuAu94UzjsQbWY6b1yu8EezFdJrJCvE++2sU/TL5oKA
RQJEYdm97KqXE0K5BYuxRKLpU03K0atfwhR2udtatz3NzyDfdyhqg1U=
-----END RSA PRIVATE KEY-----
```

认证流程对接

以 PHP 语言版本为例，对接流程如下：

1. 将 SDK 提供的库 **src/lotTid** 及其依赖 **composer.json** 整合到业务项目工程中。

2. 获取完整的设备端认证数据流，并转换为字符串形式存储至 **\$deviceData** 变量。

```
$deviceDataByteArr = [
    1, 16, 2, 16, 0, 0, 93, 25, 226, 175, 103, 16, 7, 225, 0, 41,
    176, 162, 236, 143, 158, 171, 198, 251, 171, 100, 104, 155,
    239, 44, 244, 239, 236, 156, 0, 0, 180, 208, 100, 230, 165,
    145, 135, 188, 125, 8, 130, 26, 3, 226, 129, 132, 202, 10,
    179, 14, 170, 33, 10, 116, 250, 154, 17, 34, 231, 22, 207,
    121, 240, 43, 49, 15, 46, 205, 163, 149, 155, 201, 149, 70,
    219, 190, 57, 167, 163, 19, 40, 9, 14, 157, 93, 169, 170, 57,
    75, 161, 216, 77, 208, 64, 222, 8, 87, 58, 87, 210, 239, 129,
    67, 223, 26, 78, 253, 182, 191, 46, 125, 157, 45, 8, 190, 112,
    227, 19, 145, 147, 40, 19, 38, 239, 210, 110,
];
$deviceData = "";
foreach ($deviceDataByteArr as $byte) {
    $deviceData .= chr($byte);
}
```

3. 在 [账号信息页面](#) 获取您的腾讯云主账号 ID，将其作为字符串存储至 **\$userUin** 变量。

```
$userUin = '96666666666666';
```

4. 将之前生成的私钥文件的内容作为字符串存储至 **\$userPrivateKey** 变量。

```
$userPrivateKey = "-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAl31yT12UbcVYjPtvfgO8eyi32/ETT5fgnHnbY8OzhcYW5Lh
F7V+e/efMyiiNo+XtWj38JMXMjfvRxa9VnzwKIEZC5n3wki5+qCfxlRXgR5uIO/P
5p8M/qFUqfXx5rwcH2T8q0DAWG6GbYR2iljFRprojd6x8Cb/VJllmhXMeI7jRp7r
jfuHcVhb5NIjWq+331aYITmKrdqLXE7j2XPvOBZouujjH1wE3HEWjBZOQRbNt8GC
Bgm649n06Yn6NXukUD8DNpC7eNIaCWPuzN5LRYQl1Qd7KSrjiMWgqKYmM/s/eJcV
pdKAY4l43Dsas8KYnjOm8TRxm2DZUQmcslinewlDAQABAolBAQCoAuqRuCdU8l7D
lhoSiyodeMIKwAml1N38mhOdrCMXycggdaUP9QLWXISdDxrcEn0ABO5W0qIZRuvM
E6tf7vo44Gj1y84AqwmAooW3ZwFSRkxiCjLy+POxWnbF4uqeQg70mCzkeBkl/M9F
ZM2WQ0QDC4QpsjnaFGalQtX8B1vDnCH98hEZ/WQFhkVlaeljvZqx7pDgYsP2gF7S
8xdcYDJ1Vy9Gas2vLJ8scpOmWgse6hXVvy8CDmb0tf9we3xEAH8tuI7ZvhfcizTa
0En8WJZIP9osh3zmLgyrVZ0L87iiiNgaQoVkf6q1V5LTEUOYu9q95Q7owEGHzmBo
qRQGzaQxAoGBANrt0gCwd7/VISMfOUBcf79uHjnYPX+Rbik+EjIHcXmmGuYcsj3v
t03xuaLyinNPiOmKfQ+7n/cIL8m5Bx1JYkhnd6MAS9U62xAeW5OphiTCAN4stSP1
-----"
```

```
G8r6YXCgpuwzH+5xlkjaQAIYDQpBVzdMNMMS0sPvOb6fyxD0iInku6TAoGBAPqq
YyITu4o7TpVnr9hcg3B2ehjxnOc8PTqGLC0hcNq05ZAYusd3ZALkRMVJciuwRgtK
pw9UEenCpottTPfjynKKT0AGMwdpdueZPZINi/XYrBj8z6E2Jag2adj0O1mSAay5
8DhDm20R6DsdrRskAgpH6PC7Y4tn10DIHIYVzQx5AoGBAJFsehulHNZbviYo/nGn
P+Bb+E36OtebNe7FYXbEBPeGg8Wo9xTiAPxdpsruxXflhVzvaEOw7JnSL8ZXgVx
Mozf1dN7pePNBajOUYxjhkyZbsvPV/WOBD7gnA69sZdK2/LGcX46BBYo0R+0w/A
sqXr3wKUQKm5aXJw1ouDeBFAoGACSGUOvhxj1kfQ8Hn4IVwn7kUOLkKvD+tFOg1
0Og50SNuh1b/661mz9w6zulh84qa5Gqzs5Gknd9oagtyFIMevBRIVLdlQc4Oc+Nb
JmysQizKDJeflgn+Uii6sRwq1tjliuC377Y4hUF0NESfQDCOioWZ8sFFn6AJcutr
skVVmjECgYEA0VBIt9ku5dQoDB4MwDO3ILRlf/OmkOUBsOFa06wrSoPNxlaksdCs
DUOg4DCVID62kfa6bX+o1G0+/jiPwg2q5NLICy+6Q0L/ifzgnDadYgZTPwb2ZRwM
UihxUaSdcPVrcdQ00idrwwyb3lrt0/LAGmBpMsSKelpieQWpKwqZxL8=
-----END RSA PRIVATE KEY-----";
```

5. 调用设备认证 HTTPS 接口，若调用成功，将服务端认证字节流存储至 **\$serverData** 变量，将会话 Token 存储至 **\$sessionToken** 变量，将服务端证书文件路径存储至 **\$serverCertificatePath** 变量。

```
use lotTid\Server;
$server = new Server();
$authResult = array();
try {
    $authResult = $server->authDeviceHttps($userUin, $userPrivateKey, $deviceData,
    $serverCertificatePath);
} catch (Exception $e) {
    echo "Auth device error: " . $e->getMessage() . "\n";
    throw $e;
}
$serverData = $authResult[0];
$sessionToken = $authResult[1];
```

6. 将 **\$serverData** 变量的内容作为服务端认证数据流完整传输给设备。
7. 将 **\$sessionToken** 作为该设备最新的会话 Token，按照预设的对称加密算法，使用该 Token 作为密钥进行业务数据加解密操作。