vTrust TEE UIS8850 平台 AT 指令说明书

V1.0，2023.01.04
公开

GOLDEN RIVER
谦 川 科 技

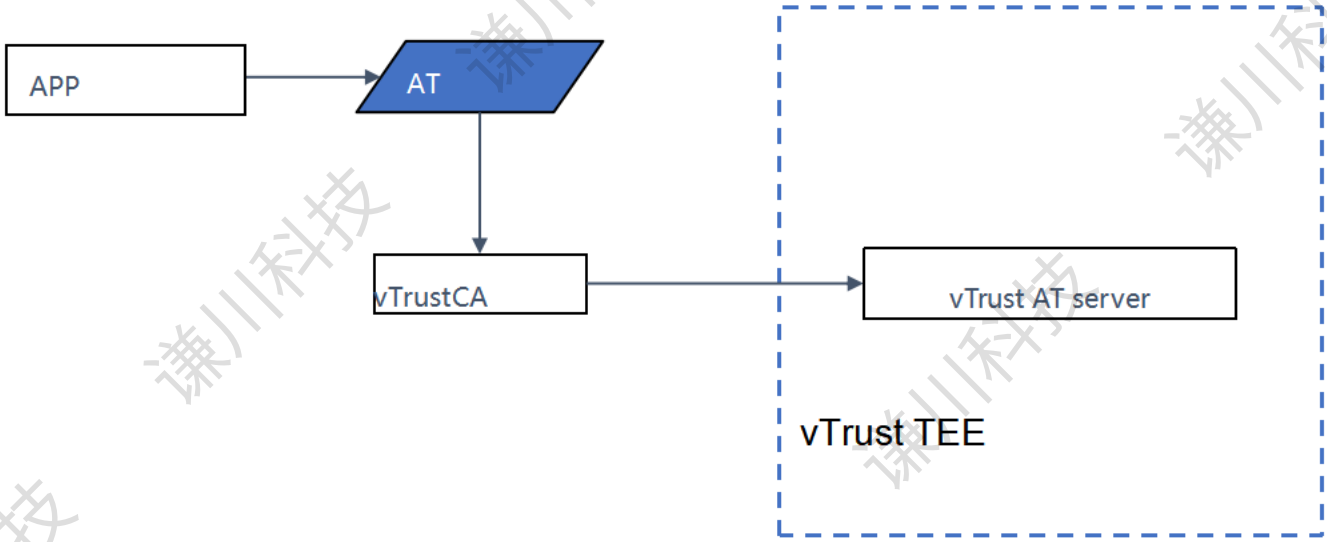
变更履历表

版本	日期	修订人	修订内容
V1.0	2022.1.4	张扬	初版

目录

1 AT 指令概要

1.1 总体结构



App 通过 AT 指令，可以快速的联系 TEE 中的 AT 指令的服务，从而方便的实现一些安全功能。

1.2 流程

Request ID 为四字节随机数，由 APP 生成。用于发送的数据与返回的数据成对匹配。每次 APP 发送数据，Request ID 不一致。UUID，由 APP 生成。

1.3 AT 命令详细

1、AT 发送指令格式

AT+VTRUST=“参数 1，参数 2，参数 3，参数 4，参数 5，参数 6，参数 7”，字符串长度

CEK 用 AES 算法对参数加密

AT 命令：1 不需要加密
其他命令都需要加密

例：AT+VTRUST=“Request ID，UUID，1”，字符串长度

AT+VTRUST=“Request ID，UUID，2，16”，字符串长度

AT+VTRUST=“Request ID，UUID，3”，字符串长度

AT+VTRUST=“Request ID，UUID，4”，字符串长度

AT+VTRUST=“Request ID，UUID，5”，字符串长度

AT+VTRUST=“Request ID，UUID，6，test，10，0，helloworld”，字符串长度

AT+VTRUST=“Request ID，UUID，7，test，10，0”，字符串长度

参数 1	参数 2	参数 3（AT 命令）	参数 4	参数 5	参数 6	参数 7
Request ID	UUID	1：获取加密密钥	不支持	不支持	不支持	不支持
Request ID	UUID	2：获取随机数	随机数长度（字节）	不支持	不支持	不支持
Request ID	UUID	3：获取 device id	不支持	不支持	不支持	不支持
Request ID	UUID	4：获取 REE 时间	不支持	不支持	不支持	不支持
Request ID	UUID	5：获取 TEE 时间（开机时间）	不支持	不支持	不支持	不支持
Request ID	UUID	6：安全存储写数据	文件名	文件中写数据大小	文件中写数据偏移量	写数据
Request ID	UUID	7：安全存储读数据	文件名	文件中读数据大小	文件中读数据偏移量	

2、AT 返回指令格式

“参数 1，参数 2，参数 3，参数 4，参数 5，参数 6”，字符串长度

CEK 用 AES 算法对参数加密

AT 命令：1 不需要加密
其他命令都需要加密

例：“Request ID，0，1，CEK 密文+CEK 密文签名”，字符串长度

“Request ID，0，2，16，随机数”，字符串长度

“Request ID，0，3，16，device id”，字符串长度

“Request ID，0，4，长度，时间”，字符串长度

“Request ID，0，5，长度，时间”，字符串长度

“Request ID，0，6”，字符串长度

“Request ID，0，7，10，10，helloworld”，字符串长度

“Request ID，-1，tee paragma error”，字符串长度

“Request ID，-1，tee get data failed”，字符串长度

参数 1	参数 2	参数 3（AT 命令）	参数 4	参数 5	参数 6
Request ID	返回值：0	1：获取加密密钥	CEK 密文+CEK 密文签名	不支持	不支持
Request ID	返回值：0	2：获取随机数	随机数长度（字节）	随机数	不支持
Request ID	返回值：0	3：获取 device id	device id 长度（字节）	device id	不支持
Request ID	返回值：0	4：获取 REE 时间	长度（字符串）	时间	不支持
Request ID	返回值：0	5：获取 TEE 时间	长度（字符串）	时间	不支持
Request ID	返回值：0	6：安全存储写数据	不支持	不支持	不支持
Request ID	返回值：0	7：安全存储读数据	文件中读出的数据长度	文件中数据总长度	数据内容
Request ID	返回值：-1	失败原因（字符串）			

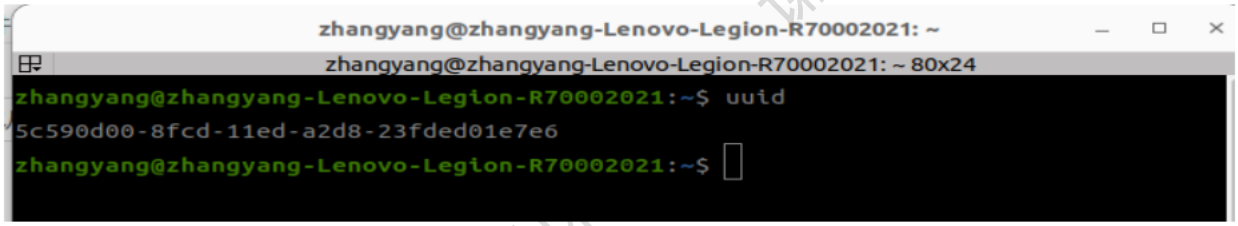
返回值：0 成功　-1 失败

3、APP 发送 AT 命令对应流程
1）APP 发送：AT+VTRUST=Request ID，UUID，1
2）CA 返回 Request ID，0，CEK 密文+CEK 密文签名
3）APP 用预置 AES 秘钥验证 HMAC 解密获得 CEK
4）APP 发送：AT+VTRUST=Request ID，UUID，CEK 加密 AT 命令数据
5）CA 返回 Request ID，0，CEK 加密 AT 命令数据
6）APP 用 CEK 解密返回数据
7）1-3 成功一次后，在使用 AT 命令重复 4-6 过程
注：AT 指令发送返回数据字符串最大长度 1024 字节

2 APP 准备

2.1 APP 预置

1.AES 密钥
2.生成 Request ID（可自行生成）：4 字节
3.UUID 用工具生成且固定值：
例：5c590d00-8fcd-11ed-a2d8-23fded01e7e6



2.2 APP AT Demo

1、Demo 文件夹

Demo 中函数已经包含加解密和数组组包可可直接使用

文件	vtrust_at_client.c

函数说明	AT 命令 1:
	static int at_get_key(uint8_t *requset_id, char *uuid, uint8_t at_command,char *out, size_t *out_size);
	AT 命令 2:
	static int at_get_rand(uint8_t *requset_id, char *uuid, uint8_t at_command,uint8_t size, char *out, size_t *out_size;)
	AT 命令 3:
	static int at_get_device_id(uint8_t *requset_id, char *uuid, uint8_t at_command,char *out, size_t *out_size);
	AT 命令 4:
	static int at_get_ree_time(uint8_t *requset_id, char *uuid, uint8_t at_command,
	char *out, size_t *out_size)
	AT 命令 5:
	static int at_get_tee_time(uint8_t *requset_id, char *uuid, uint8_t at_command,
	char *out, size_t *out_size)
	AT 命令 6:
	static int at_storge_write(uint8_t *requset_id, char *uuid, uint8_t at_command,
	char *name, size_t size, size_t offset, char *data,
	char *out, size_t *out_size)
	AT 命令 7:
	static int at_storge_read(uint8_t *requset_id, char *uuid, uint8_t at_command,
	char *name, size_t size, size_t offset, char *out,
	size_t *out_size)
	接收数据处理:
	static int at_data_process(char *recv_msg, size_t msg_size, void *out,size_t *out_size)
	app at 命令发送接收消息函数
	int app_at_msg(uint8_t at_command, void *in, size_t in_size, void *out,size_t *out_size)
	app at 命令测试函数
	void vtrust_at(void)

文件	at_aes_crypt.c

函数说明	aes 加密：
	int at_aes_encrypt(const struct key *key, at_crypt_data *data_in,at_crypt_data *data_out, const struct iv *iv_in);
	aes 解密:
	int at_aes_decrypt(const struct key *key, at_crypt_data *data_in,at_crypt_data *data_out, const struct iv *iv_in);
	aes 加密+hmac：
	int at_aes_hmac_encrypt(const struct key *key, at_crypt_data *data_in,at_crypt_data *data_out, const struct iv *iv_in);
	aes 解密+验证 hmac：
	int at_aes_hmac_decrypt(const struct key *key, at_crypt_data *data_in,at_crypt_data *data_out, const struct iv *iv_in);

文件	at_base64url.c

函数说明	base64url 编码：
	int at_base64url_encode(const uint8_t *input, size_t inlen, char **output, size_t *outlen);
	base64url 解码：
	int at_base64url_decode(const char *input, size_t inlen, uint8_t **output, size_t *outlen);

2、使用 Demo 修改 vtrust_at_client.c

修改 request_id 和 app_uuid

注:aes_key 和 iv_temp 不可修改

```
23
24  static char app_uuid[40] = {"186b54c8-8bd0-11ed-a8b1-cb652ea15773"};
25
26  static uint8_t request_id[4] = {0x12, 0x43, 0x55, 0x78};
27
28  static struct key cek = {0};
29
30  struct iv iv_temp = {
31      .byte = {1, 67, 43, 87, 46, 67, 34, 56, 78, 97, 56, 78, 24, 16, 45, 73},
32  };
33  struct key aes_key = {
34      .byte = {20, 47, 89, 56, 23, 47, 89, 52, 12, 47, 89, 56, 123, 7, 85, 46},
35  };
36
```

用 APP 内数据发送和接收函数替换 at_send_recv

```c
static int at_send_recv(void *in, size_t in_size, void *out, size_t *out_size) {
  int rc;

  nbl_chn_t *chn;
  char *srv_name = AT_SRV_NAME;

  if ((!in) || (!out) || (in_size > MSG_SIZE_MAX)) {
    rc = NBL_ERR_INVALID_ARGS;
    goto failed;
  }
  rc = nbl_chn_open(srv_name, 0, &chn);
  if (rc != NBL_NO_ERROR) {
    NBL_LOGE("Connect to %s failed.\n", srv_name);
    rc = NBL_ERR_IO;
    goto connect_failed;
  }

  rc = nbl_chn_write_buf(chn, in, in_size);
  if (rc != in_size) {
    NBL_LOGE("Service B Client write buffer failed (%d).\n", rc);
    rc = NBL_ERR_IO;
    goto write_buf_failed;
  }
  rc = nbl_chn_read_buf(chn, out, *out_size, NBL_TIMEOUT_INFINITY);
  if (rc > *out_size) {
    rc = NBL_ERR_IO;
    goto read_buf_failed;
  }
  *out_size = rc;
  NBL_LOGI("buf size: %zu read buf:%s\n", *out_size, out);
  rc = NBL_NO_ERROR;
read_buf_failed:
write_buf_failed:
  nbl_chn_close(chn);
connect_failed:
failed:
  return rc;
}
```

参考或者修改 app_at_msg 可发送接收 AT 指令

```
int app_at_msg(uint8_t at_command, void *in, size_t in_size, void *out,
               size_t *out_size) {
  int rc = 0;
  memset(in, 0, in_size);
  memset(out, 0, *out_size);
  switch (at_command) {
  case AT_COMMAND_GET_KEY:
    rc = at_get_key(request_id, app_uuid, AT_COMMAND_GET_KEY, in, &in_size);
    break;
  case AT_COMMAND_GET_RAND:
    rc = at_get_rand(request_id, app_uuid, AT_COMMAND_GET_RAND, 16, in,
                     &in_size);
    break;
  case AT_COMMAND_GET_DEVICE_ID:
    rc = at_get_device_id(request_id, app_uuid, AT_COMMAND_GET_DEVICE_ID, in,
                          &in_size);
    break;
  case AT_COMMAND_GET_REE_TIME:
    rc = at_get_ree_time(request_id, app_uuid, AT_COMMAND_GET_REE_TIME, in,
                         &in_size);
    break;
  case AT_COMMAND_GET_TEE_TIME:
    rc = at_get_tee_time(request_id, app_uuid, AT_COMMAND_GET_TEE_TIME, in,
                         &in_size);
    break;
  case AT_COMMAND_STORAGE_WRITE_FILE:
    rc = at_storge_write(request_id, app_uuid, AT_COMMAND_STORAGE_WRITE_FILE,
                         "test", strlen("nihao"), 0, "nihao", in, &in_size);
    break;
  case AT_COMMAND_STORAGE_READ_FILE:
    rc = at_storge_read(request_id, app_uuid, AT_COMMAND_STORAGE_READ_FILE,
                        "test", strlen("nihao"), 0, in, &in_size);
    break;
  default:
    sprintf(out, "%s", "app no find at command");
    *out_size = strlen(out);
    return -1;
    break;
  }

  if (rc < 0) {
    return rc;
  }
  NBL_LOGI("data size: %zu,in data:%s", in_size, in);
  char msg_recv[1024];
  size_t mgs_recv_size = 1024;
  memset(msg_recv, 0, mgs_recv_size);
  rc = at_send_recv(in, in_size, msg_recv, &mgs_recv_size);
  if (rc < 0) {
    sprintf(out, "%s", "at send recv falied");
    *out_size = strlen(out);
    return rc;
  }
  rc = at_data_process(msg_recv, mgs_recv_size, out, out_size);
  if (rc < 0) {
    return rc;
  }
  return rc;
}
```

static int at_data_process(char *recv_msg, size_t msg_size, void *out,size_t *out_size)

说明：

recv_msg：接收数据

msg_size:接收数据大小

out:解析后的数据

out_size:out 的大小

out	out_size 值	返回值
1：获取加密密钥	cek 长度：16 字节	cek
2：获取随机数	随机数长度	随机数
3：获取 device id	Device id 长度：16 字节	device id
4：获取 REE 时间	时间长度：8 字节	前 4 字节：秒 后 4 字节：毫秒
5：获取 TEE 时间	时间长度：8 字节	前 4 字节：秒 后 4 字节：毫秒
6：安全存储写数据	0	无
7：安全存储读数据	文件参数大小	0-3 字节：读数据长度 4-7 字节：文件总数据长度 剩下字节：读数据内容

2、lib 文件夹

at_crypt 文件编译出 libat_crypt.a 可直接使用 libat_crypt.a

3、include 文件夹

Demo 使用的头文件