

# 乐谱加密前端部署说明\_5.8更新版

Author: 郭辉铭

## 一. 更新说明

由于曲谱部分模型较多，对单个模型分别进行加密解密，请求授权耗时巨大，因此合并运行，减少运行时间成本！


主要做了以下改进：

- 不再对模型单独加解密，而是以功能为单位，即用户某段时间调用一个功能（比如图片拍摄功能）时进行一次授权即可；
- 对单个功能，不需要多个私钥和公钥。而是只需要一个即可：生成公私钥部分代码单独分离并修改调用流程；
- 授权服务器传给后端的每个encodedInfo不再只包含一个模型的相关信息，而是嵌套的包含多个模型。因此解析的时候，需要分别读取。之前该部分是在算法推理中完成，而现在需要前端配合，在当前模型执行推理的上一层完成该项任务。

**前端调用只关注标题三内容即可**

## 二. 第二步加密流程叙述


### step1: 模型拆解

 算法模型model拆分成两个部分：model\_p1(提交前端)以及model\_p2（大致5kb提交后端传给授权服务器），授权服务器会给模型对应的salt\_key。

涉及代码：

file\_split.cpp中的file\_string（）用于模型拆分：输入整个model文件，输出两个拆分后的model文件

### step2: 公私钥以及sign值的生成

-  step2\_1 当需要调用某个功能时，而该功能又包含了多种模型。此时前端向算法请求获得一对公私钥。公钥进行base64加密得到base64\_pu，私钥留待模型解密时使用；
- step2\_2 针对单个模型，base64\_pu+ "&" +该模型对应的salt生成MD5。

- 将单个模型对应的base64\_pu以及MD5返回给前端。
- 前端传送给授权服务器并认证之。

涉及代码：

2\_1) 生成公私钥并对公钥进行base64编码：encode\_decode.cpp---\_getBase64\_pubKey();  
返回string类型：base64\_pubKey(step2\_2要用), 并得到唯一的私钥myprivatekey。

2\_2) 针对单个模型使用base64编码后的字符串与对应的salt值生成MD5：encode\_decode.cpp--  
-- \_getPubkey\_singleModel()

输入：模型名称，2\_1的返回值

输出：sign值，传给授权服务器

### step3: 授权服务器认证并处理



- 授权服务器认证通过之后，服务器生成aes密钥，用以加密model\_p2以及md5\_key和authorization\_id,并用base\_64编码生成enc\_aes\_data;
- 用step2中生成的公钥加密aes密钥并用base\_64编码获得en\_params\_aes\_key。
- 两者合二为一作为json字典传送至后端

以上在授权服务器完成，再由前端传给算法端。

1) 授权服务器传给前端的是：encodedInfo。

2) 前端和后端处理成string字符传给算法（同以前）

### step4: encodedInfo信息分离并解密出model\_p2



算法端对encodedInfo字符进行处理，分离出en\_params\_aes\_key和enc\_aes\_data\_dict

涉及代码：

- 将encodedInfo按照模型分离：encode\_decode.cpp----->json\_readFromStr();得到key=model\_name, value=enc\_aes\_data。而en\_params\_aes\_key则另外存储。因为多模型共用一个en\_params\_aes\_key。
- 对单个模型进行模型解密encode\_decode.cpp----->decode\_authorization()。返回单个模型的model\_p2。多个组合成字典，因为是一次解密，分别传参至算法。

代码解释（其他人员可不看）

授权服务器传过来的json文件处理：encode\_decode.cpp-----decode\_authorization();

1. 获取en\_params\_aes\_key的base64解密结果key\_decoded

```
string key_decoded =  
base64_decode(encode_info_json["en_params_aes_key"].asString());
```

2. 应用step2中的私钥进行解密：

```
int key_length = private_decrypt((unsigned char*)key_decoded.c_str(),  
key_decoded.length(), (unsigned char*)myprivatekey.c_str(), aes_keys);
```

获取结果：aes\_keys---user\_key---decrypt\_key

3. 再次进行base64解密获得待拼接的临时字符串temp\_model\_p2，可以存储进字典中：

```
temp_model_p2 = base64_decode(decode_result["params"].asString());
```

## step5: 模型合并

🏐 解密后的model\_p2与前端传进的model\_p1进行组合，生成新的buffer。之后进行模型推理模块。

该部分在算法推理模块内执行

## 三. 前端调用接口

前端只参与step2以及step4。step5注意传参格式

### 1) 调用step2

依次调用encode\_decode.cpp中的\_getBase64\_pubKey()和\_getPubkey\_singleModel();

1. 当前功能调用一次encode\_decode.cpp--->\_getBase64\_pubKey();

输入：无

输出：base64编码后的公钥（返回值）。以及私钥myprivatekey(cpp内全局变量)

2. 对每个模型均调用一次encode\_decode.cpp---->\_getPubkey\_singleModel(string model\_name, string base64\_pubKey);

输入：model\_name==当前模型名称；base64\_pubKey==1.中的返回值，所有模型共用一个。此处为方便，可以修改作用域

返回：每个模型生成对应的sign值resultStr

step2\_2的返回值传至授权服务器即可。

模型名称须一一对应：

曲谱页面框选："music\_page\_frame"

曲谱正反判断："updown\_pruned\_v2"

曲谱页面检测: "music\_det"

曲谱中文识别: "music\_rec\_ch"

曲谱外文识别: "music\_rec\_fri"

## 2) 调用step4

只需要调用encode\_decode.cpp--->get\_decodeModel\_p2(string encodedInfo, vector<string> function)即可:

输入: encodedInfo==授权服务器认证通过后的返回值, 需要处理成字符串传递; function = {"music\_det", "music\_rec\_ch", "music\_rec\_fri", "updown\_pruned\_v2"};

返回: model\_p2\_sets=={key: value}其中key==模型名称, value==该模型名称下对应的解密后model\_p2字符串。

## 3) 模型合并 (算法内部, 前端关注传参形式即可)

由算法端调用modelCombine()

2) 中返回的是该功能下所有模型的部分字符组成的字典。模型推理时, 前端需要单独传递该模型的model\_p2