密码学大作业

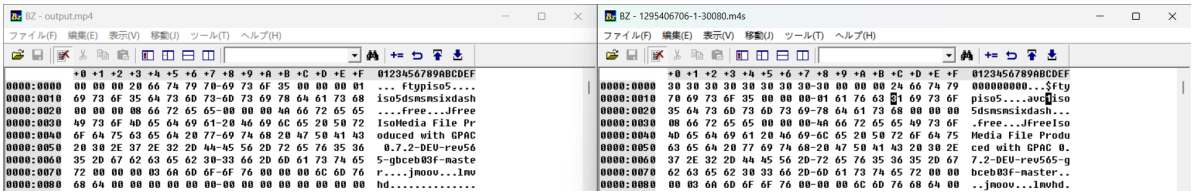
——对Bilibili客户端缓存视频/音频的解码，以及对Bilibili网页端登录流程的分析

1. 对客户端缓存视频/音频的解码

通过B站客户端下载的视频文件被分为**视频**和**音频**两部分，且经过加密后保存为.m4s格式的文件

上图中的"30080.m4s"为加密后的视频文件，"30280.m4s"为加密后的音频文件，此时两个文件无法直接打开。

为了分析加密的原理，我下载了一个辅助软件将加密后视频文件解密，并分析解密前后二者的二进制文件区别。



可以看到二者的头部有着明显的区别。因为视频播放需要较好的实时性，所以我猜测此处的加密只是在原有二进制序列的基础上做了一点小的改动而不是整体加密，比如增加一些序列。

认真观察之后，我发现加密后的二进制序列翻译出来的结果比加密前多出了九个“0”、序列“avc1”，并将原来的一个空格换成了'$'，所以根据这一特点，我编写了以下程序用来解密视频文件和音频文件

```python
def fix_m4s(target_path: str, output_path: str, bufsize: int = 256*1024*1024) ->
None:
    assert bufsize > 0
    with open(target_path, 'rb') as target_file:
        header = target_file.read(32)
        new_header = header.replace(b'000000000', b'')
        new_header = new_header.replace(b'$', b' ')
        new_header = new_header.replace(b'avc1', b'')
        with open(output_path, 'wb') as output_file:
            output_file.write(new_header)
            i = target_file.read(bufsize)
            while i:
                output_file.write(i)
                i = target_file.read(bufsize)
```

其中参数解释如下：

- target_path：待解码的视频/音频文件路径
- output_path：解码后输出的视频/音频文件路径
- bufsize：缓冲区大小，用于防止输入文件过大导致溢出

在对以上两个文件解密后，我得到了两个可以播放的视频文件和音频文件：

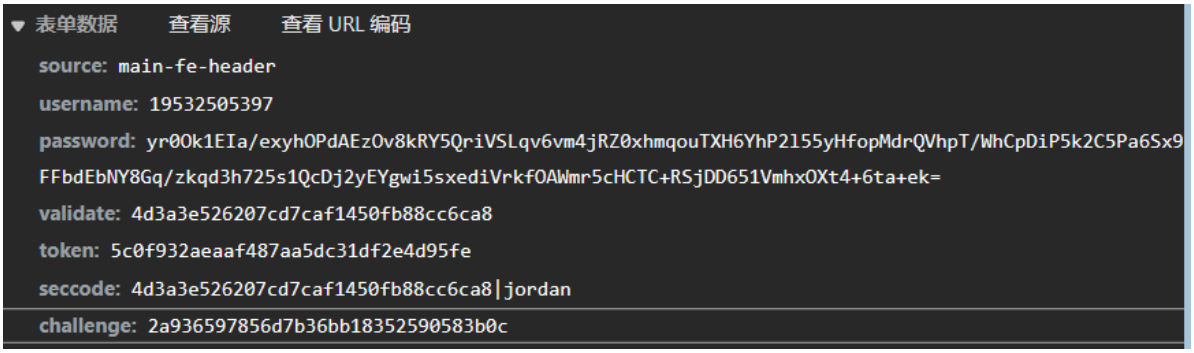
至此，客户端缓存内容的解密结束

2. 网页端登录认证流程分析

声明：本节的代码分析都体现在注释中！

要分析Bilibili网页端的登录认证流程，首先要知道在登陆过程中都使用了哪些信息

首先使用错误的密码进行登录，获取反馈信息



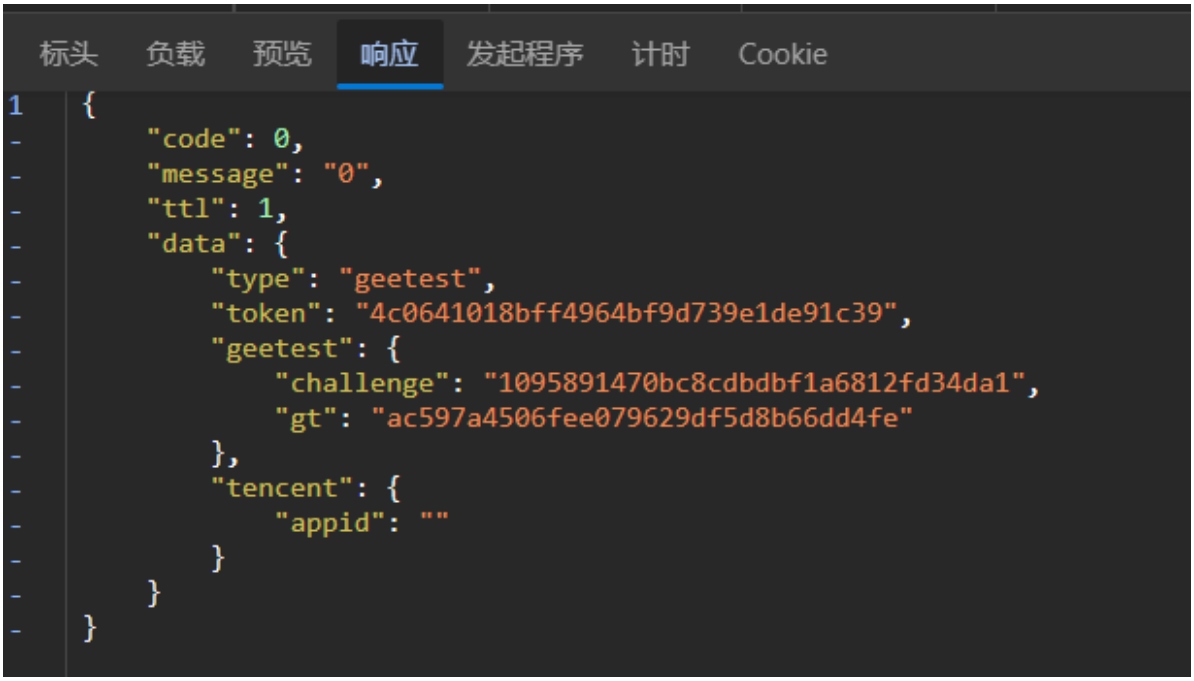
其中：

- $username$：登陆的账号
- $password$：加密后的密码，且从最后的'='可以推测，这里的password是加密后通过base64编码得到的
- 剩下的四个值$validate, token, seccode, challenge$便是加密参数

2.1 人机验证

Bilibili网页版的登录有三种方式：账号密码登录、短信验证码登录和扫码登陆。其中扫码登陆不需要人机验证，下面来分析账号密码登录过程中的人机验证

首先请求验证码参数，得到登陆密钥

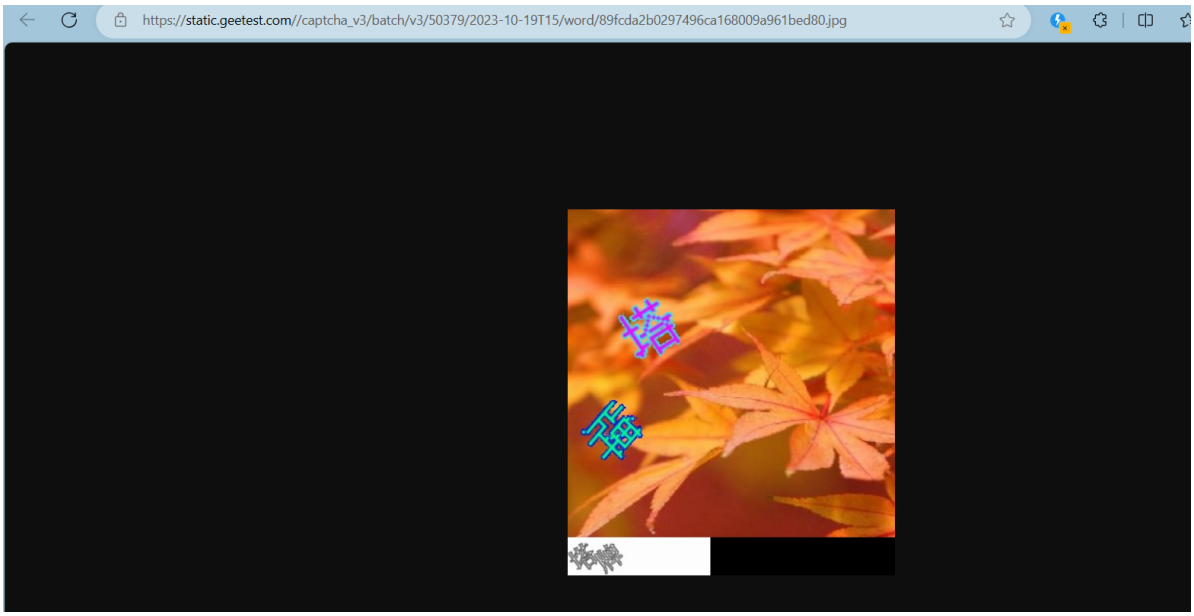


内容分析：

- $code$：返回值0代表成功
- $message$：返回的信息
- ttl：固定内容
- $data$：信息本体
  - $geetest$：极验captcha数据
    - gt：极验id，一般为固定值
    - $challenge$：极验KEY，由B站后端产生，用于人机验证
  - $token$：登录API token
  - $type$：验证方式，这里为"geetest"极验

然后获取图片验证码并填写，得到$validate$

这里将$image_server$中的任意一个字符串和pic字符串相连接都可以得到验证码的图片地址，例如：



在按照要求正确点击图片验证码后，会收到一个$validate$

人机验证到此结束

2.2 密码加密

浏览器向服务器发送获取公钥和盐的GET请求，得到以下内容：

其中：

- $hash$：密码盐值，有效时间为20s，长度恒为16个字符，需要拼接在明文密码之前
- key：RSA公钥，这里显示为PEM编码格式，后续加密密码时要使用

加密操作：$encoded\_password = RSA(key, hash + password)$

加密之后还要对密文进行base64编码操作，得到$encoded_password_base64$

对加密过程的分析如下：

通过在开发者工具进行全局扫描"encrypt"这一关键词，我将分析目标定位到了两个.js文件中的代码：

- $miniLogin.umd.min.1.js$：各种加密算法、数据位操作的实现
- $miniLogin.umd.min.js$：对密码、token等信息的验证

下面这段代码摘自$miniLogin.umd.min.1.js$

```
                , Q = function() {     //该函数返回一个RSA算法的实现
                  function t() {       //构造函数
                      this.n = null,
                      this.e = 0,
                      this.d = null,
                      this.p = null,
                      this.q = null,
                      this.dmp1 = null,
                      this.dmq1 = null,
                      this.coeff = null
                  }
                  return t.prototype.doPublic = function(t) { //使用公钥对给定的参数t
进行加密操作，返回加密后的结果
                      return t.modPowInt(this.e, this.n)
                  },
                  t.prototype.doPrivate = function(t) {       //使用私钥对给定的参数t
进行解密操作，返回解密后的结果
                      略
                  },
                  t.prototype.setPublic = function(t, e) {
                      略        //设置公钥，参数t是公钥的十六进制表示形式，参数e是公钥指数的十
六进制表示形式
                  },
                  t.prototype.encrypt = function(t) {
                      略        //使用公钥对给定的参数t进行加密操作，返回加密后的结果的十六进
制表示形式
                  },
                  t.prototype.setPrivate = function(t, e, i) {
                      略        //设置私钥，参数t是私钥的十六进制表示形式，参数e是公钥指数的十
六进制表示形式，参数i是私钥的十六                                    进制表示形式。
                  },
                  t.prototype.setPrivateEx = function(t, e, i, r, n, s, o, h) {
```

```
                    略        //设置私钥的扩展参数，参数t、e、i、r、n、s、o、h分别表示私钥、公钥指数、私钥参数p、私钥参数                                                q、私钥参数dmp1、私钥参数dmq1、私钥参数coeff的十六进制表示形式
                }。
                t.prototype.generate = function(t, e) {
                    略        //生成指定位数的公私钥对，参数t是位数，参数e是公钥指数的十六进制表示形式。
                },
                t.prototype.decrypt = function(t) {
                    略        //使用私钥对给定的参数t进行解密操作，返回解密后的结果
                },
                t.prototype.generateAsync = function(t, e, i) {
                    略        //异步方式生成公私钥对，参数t是位数，参数e是公钥指数的十六进制表示形式，参数i是回调函数，在生成                                                完成后调用该函数。
            },
                t.prototype.sign = function(t, e, i) {
                    略        //使用私钥对给定的参数t进行签名操作，参数e是一个哈希函数，参数i表示哈希函数的索引，返回签名后的                                                结果的十六进制表示形式
                },
                t.prototype.verify = function(t, e, i) {
                    略        //使用公钥对给定的参数t和签名e进行验证操作，参数i是一个哈希函数，返回验证结果
                },
                t
            }()
```

上面这段代码是$password$加密过程中使用的RSA算法的实现（为了节省篇幅，省去了函数的具体实现细节）

下面这段代码摘自$miniLogin.umd.min.js$

```
lt = function(t) {
   //创建Promise对象，内部包含一个执行器函数
   return new Promise(function() {
       var e = Y(Q().mark((function e(r) {       //标记异步操作
             var o, i, a, c;
             return Q().wrap((function(e) {
                while (1)
                   switch (e.prev = e.next) {
                      case 0:
                         return e.next = 2,
                         ut();
                      case 2:
                          return o = e.sent,
                          e.next = 5,
                          // 通过动态导入的方式加载一个文件
                          n.e(1).then(n.t.bind(null, "24e5", 7));
                      case 5:
                           i = e.sent,
                           // 获取JSEncrypt对象用于RSA加密操作
                           a = i.JSEncrypt,
                           // 若code为0，即正常，则表示成功获取到了公钥
                           0 === o.code ? (c = new a,
                           //设置公钥，从o.data.key获取公钥数据
                           c.setPublicKey(o.data.key),
```

```
                        //进行加密操作
                        r(c.encrypt(o.data.hash + t))) : r("");
                    case 8:
                        case "end":
                        return e.stop()
                    }
            }
        ), e)
    }
))));
    // 加密结果作为参数传递给回调函数r，该回调函数是异步函数的参数
    // 当整个异步函数执行完毕后。Promise对象的状态将被解析，并将加密结果作为解析值
    return function(t) {
        return e.apply(this, arguments)
            }
        }())
    }
```

上面这段代码实现了对password进行加密的操作

2.3 开始登陆

在这一步中根据上面得到的信息进行登录，若登陆成功则得到一个cookie值

$$cookie = Login(username, encoded\_password\_base64, token, challenge, validate)$$

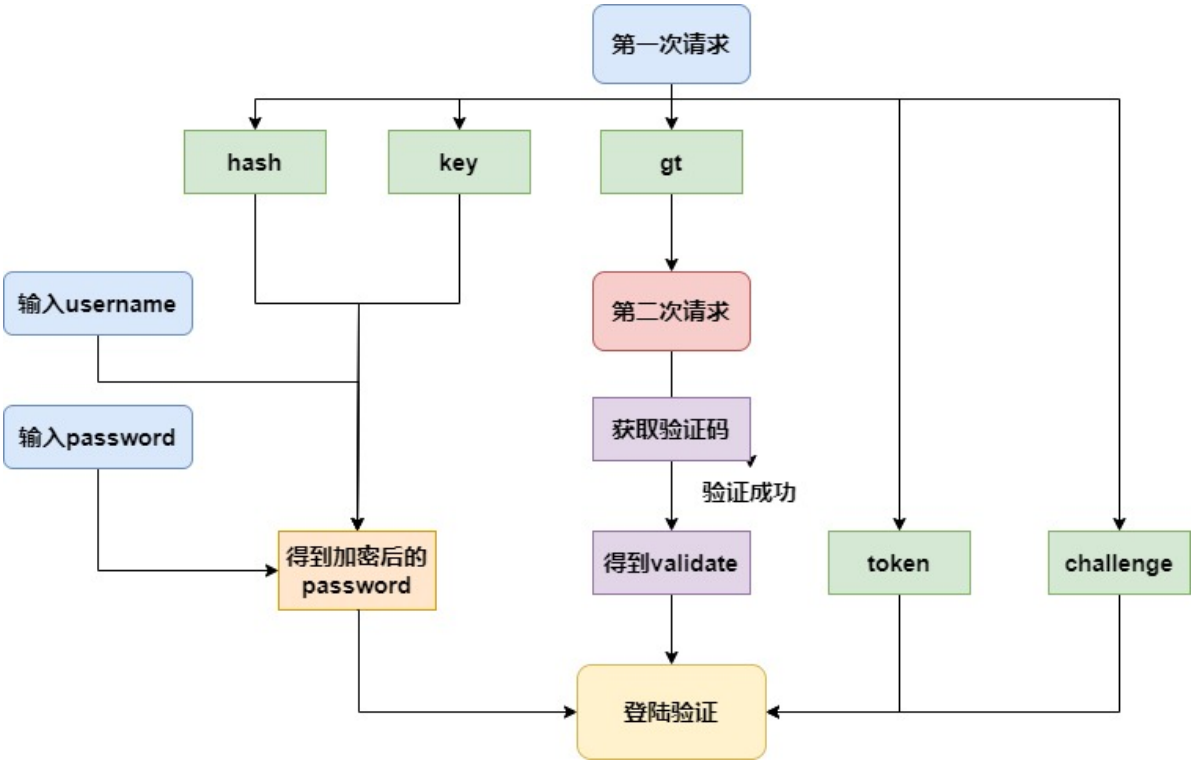
登陆成功后会返回信息，其中包含时间戳等

```
handlerLogin: function (t) {
    var e = this;
    return de(fe().mark((function n() {     //创建异步函数
        var r, o;
        return fe().wrap((function (n) {    //包装一个生成器函数
            while (1)
                switch (n.prev = n.next) {
                case 0:
                    return n.next = 2,
                    lt(e.subForm.password); //利用RSA算法加密password
                case 2:
                    r = n.sent,             //获取加密结果
                    o = {                   //初始化一部分登陆属性
                          source: e.$origin,
                          username: e.subForm.username,
                          password: r
                        },
                    // 根据验证码方式来判断该如何初始化剩余的登陆属性
                    o = "geetest" === t.captcha_type ? Object.assign(o, {
                            alidate: t.validate,
                            token: t.token,
                            seccode: t.seccode,
                            challenge: t.challenge
                        }) : Object.assign(o, {
                                captcha: t.img_code,
                                token: t.token
                            }),
                    tt(o).then((function (t) {
```

```
                        if (t.code === ft) {    //验证是否登陆成功
                              e.isClick = !1;
                               var n = t.data
                                , r = n.status
                                , o = n.url;
                                0 !== r ? (e.mini_info.electornPc_jump = o,
                                // 登陆成功则发送消息，其中包含更新后的token和时间戳
                                window.location.href = o) :
e.$emit("loginSuccess", {
                                      type: "pwd",
                                      refresh_token: t.data.refresh_token,
                                      timestamp: t.data.timestamp
                                   })
                                } else        //登陆失败
                                      e.isClick = !1,
                                      e.$toast.info(t.message).then((function
() {
                                      [86667, 86669, 86670].includes(t.code) &&
e.gotoSms(!1)
                                   }
                              ))
                           }
                     ));
               case 6:
               case "end":
                     return n.stop()
                  }
               }
            ), n)
         }
      ))()
   }
```

以上便是登录认证的流程

流程总结如下：


第一次请求
hash
key
gt
输入username
第二次请求
输入password
获取验证码
验证成功
得到加密后的
password
得到validate
token
challenge
登陆验证