Web视频播放与数据安全

Created and last modified by 陈明成 on Aug 06, 2021

HTML5 的 video 标签的安全性

H5的video/audio标签可以方便的播放音视频，只需要把资源的链接设置为src属性即可。但是这种方式也把资源的链接暴露了出来，用户可以随意下载、播放音视频数据。那么如何提高媒体资源的安全性呢?

保证媒体资源的数据安全

Referer 防盗链

浏览器内部去下载视频数据，归根到底是发起了一个HTTP请求，那么就可以围绕HTTP协议请求做一些功课。

Referer 是HTTP请求 header 的一部分，浏览器向web服务器发送请求的时候会在header中带上当前的地址，那么服务端可以通过黑白名单控制哪些地址的网页可以播放这个视频。目前云存储服务商大部分都支持referer防盗链功能。

但是客户端可以伪造这个值，所以Referer不是绝对可信的，只能作为辅助手段。

Blob URL

通俗的讲 Blob URL 也是一种URL，和一般的http开头的URL相比，它是blob:// 开头的。它是浏览器维护在内存的一堆二进制数据，是一种本地的URL。即便用户拿到了这个URL，也没办法下载和播放。

- 关于Blob URL的API

```
// 创建
objectURL = URL.createObjectURL(object); // 参数类型为 File、Blob 或者 MediaSource

// 回收
URL.revokeObjectURL(objectURL);
```

有了Blob URL，就可以通过HTTP先把视频下载到浏览器本地，然后转为Blob URL使用。然而视频下载的过程依然会暴露视频文件的真实地址，那么如何进一步提高安全性呢?

视频文件加密

如果视频文件是加密存储的，那么就算有人下载了视频，甚至攻入服务器拿到了视频文件，无法正常播放也是徒劳的。

视频文件加密算法选择

加密算法主要分为两大类，一种是可逆加密算法、一种是不可逆加密算法，可不可逆的区别是能否从加密后的密文还原成明文，能还原的就是可逆算法，否则就是不可逆。而对于可逆性算法又分为对称性加密和非对称性加密，区别的关键在于加密和解密的密钥是否相同。

哈希/摘要

常用的哈希算法有MD5、SHA系列等等，特点是不可逆、数据完整性敏感

对称性加密

主要有DES、3DES、AES等标准，AES是DES的替代，现在常用的是AES。

非对称性加密

主要使用的是RSA，有两种密钥，公钥是加密用的，私钥是解密用的。

对于加密大的文件而言适合选择AES加密。

AES 加密

密钥

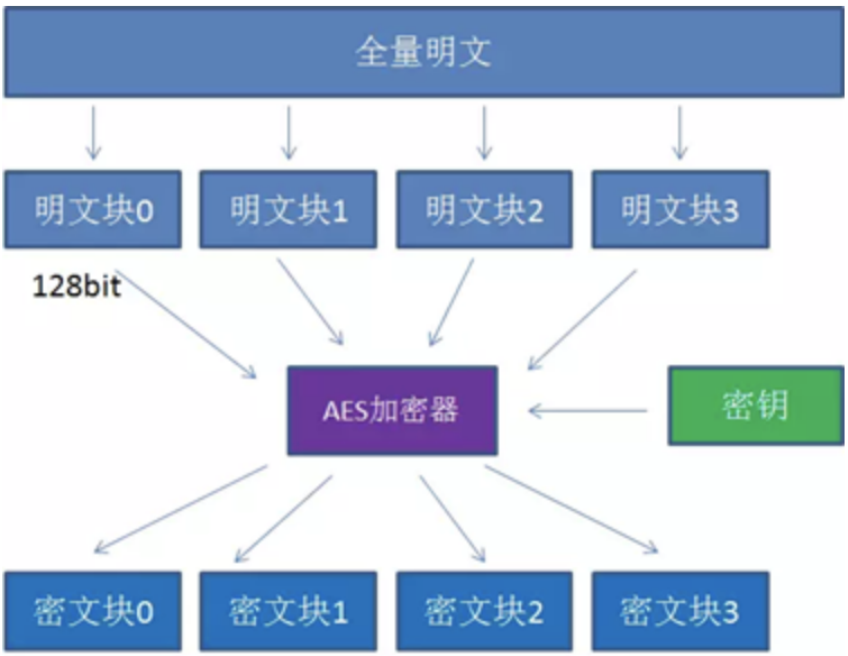
AES128、AES192、AES256， 数字对应密钥的长度（bit）

密钥可随机生成，解密时必须使用加密的密钥才能成功解密

填充

填充方式是AES的一个重要参数，加密和解密时填充方式必须一致。

要想了解填充的概念，我们先要了解AES的**分组加密**特性。



先把全量的明文分成一个个独立的明文块，一块的长度和密钥的长度一致。再把每一块使用密钥再通过加密器的复杂处理转为密文块，再把所有的密文块拼接在一起，就形成了全量的密文。

但是这里设计到一个问题，以AES128为例，如果最后一块明文块不足128bit怎么办，就需要对其进行**填充**（padding）。

- NoPadding，不进行填充，但是需要保证全量明文长度是128bit的整数倍
- PKCS5/PKCS7 Padding（默认），若明文块不足16个字节（128bit），则在末尾补足相应的字符数，每个字符为缺少的字节数比如 {1,2,3,4,5,a,b,c,d,e},缺少6个字节，则补全为{1,2,3,4,5,a,b,c,d,e,6,6,6,6,6,6}
- ISO10126Padding，和PKCS5Padding类似，只不过只有最后一个字符为缺少的字符数，其他字符为随机填充

需要注意的是，加密和解密需要使用同一种填充方式

模式

- ECB　每一个明文块独立处理；有利于并行计算，但是相同的明文会加密出相同的密文
- CBC　引入初始偏移量IV，每组的密文作为下一组的偏移量，防止相同的明文块加密成相同的密文块
- CTR
- CFB
- OFB

crypto-js

> crypto-js 是一个纯 javascript 写的加密算法类库 ， 可以非常方便地在 javascript 进行 MD5、SHA1、SHA2、SHA3、RIPEMD-160 哈希散列，进行 AES、DES、Rabbit、RC4、Triple DES 加解密

下面是利用crypto-js进行AES加解密代码。使用了CBC模式，填充方式为PKCS7，并把密钥的前四个字节作为了偏移量IV参数。

```
// 加密
export const encode = (key, content) => {
  const encodeKey = CryptoJs.enc.Base64.parse(key)
  const ivWords = [...encodeKey.words].splice(0, 4)
  const iv = CryptoJs.lib.WordArray.create(ivWords)
  const res = AES.encrypt(content, encodeKey, {
    iv: iv,
    mode: CryptoJs.mode.CBC,
    padding: CryptoJs.pad.Pkcs7
  }).ciphertext;
  return res;
};
```

```
// 解密
export const decode = (key: string, content: string | WordArray) => {
  const decodeKey = CryptoJs.enc.Base64.parse(key)
  const ivWords = [...decodeKey.words].splice(0, 4)
  const iv = CryptoJs.lib.WordArray.create(ivWords)
  var res = AES.decrypt(content, decodeKey, {
    iv: iv,
    mode: CryptoJs.mode.CBC,
    padding: CryptoJs.pad.Pkcs7
  })
  return WordArrayToArrayBuffer(res);
 }
```

流媒体播放

上面介绍了通过BlobUrl，以及文件资源加解密等方式来进行web视频播放，保证远程资源的数据安全。但是目前的架构依然存在一些问题

- 需要对整个文件下载完毕后转成BlobUrl
- 只能满足一次播放整个曲目的需要，无法 拆分/合并 多个缓冲文件
- 流媒体在MSE之前还在使用 Flash 进行服务，以及通过 RTMP 协议进行视频串流的 Flash 媒体服务器

MSE，让浏览器播放器进入H5时代！

有了MSE（Media Source Extensions）后就不一样了，允许我们生成一个指向 MediaSource 对象的BlobUrl，MediaSource是一个包含将要播放的媒体文件以及准备信息的容器，以及可以通过 SourceBuffer 进行二进制缓冲文件的拼接。

MSE包含多个复杂的class，其中有两个比较重要的API

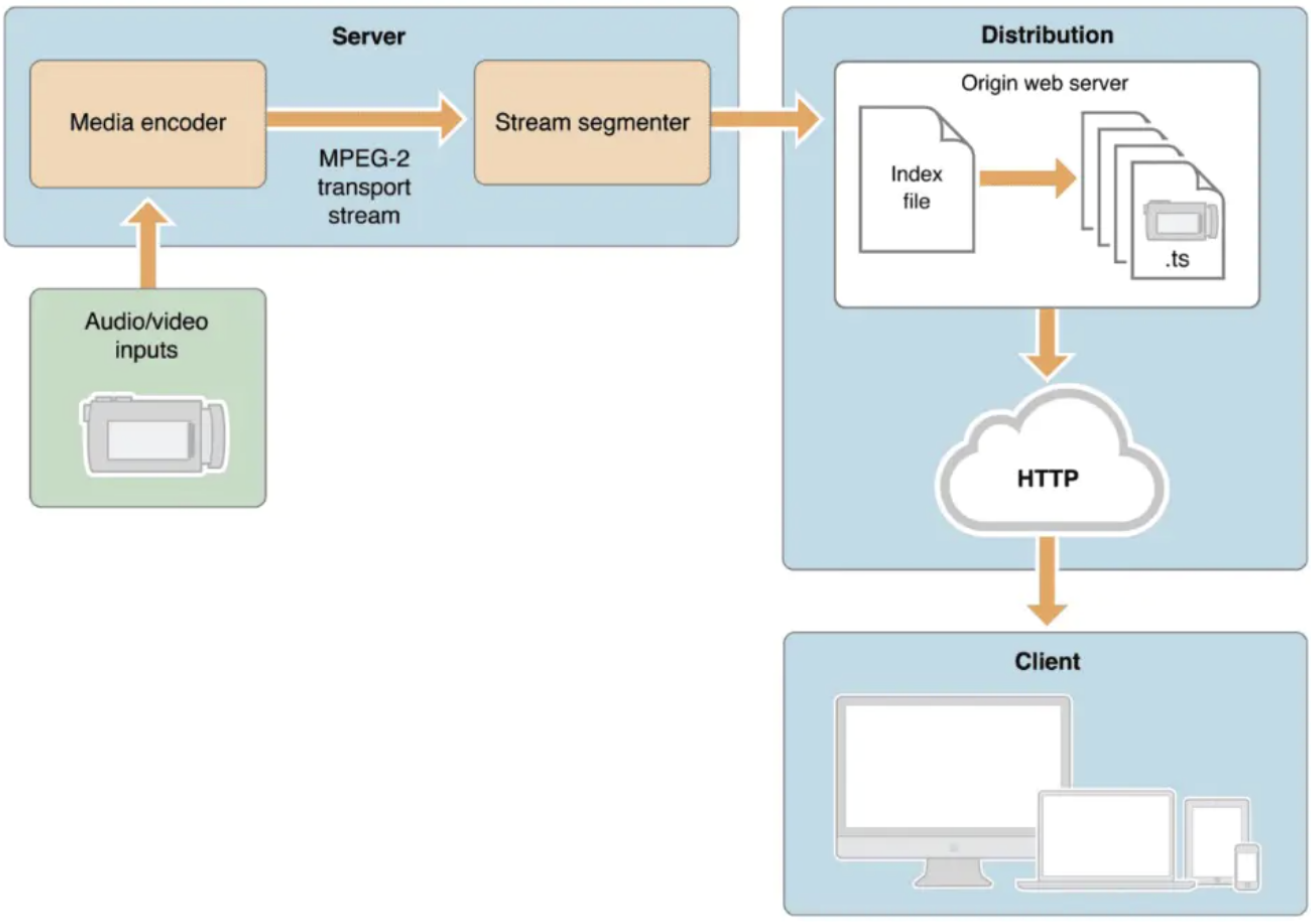
```
// 创建一个带有给定MIME类型的新的 SourceBuffer
MediaSource.addSourceBuffer(mime)

// 向SourceBuffer中添加媒体二进制数据（ArrayBuffer）
SourceBuffer.appendBuffer()
```

那么利用SourceBuffer.appendBuffer()就可以不断更新视频资源，同时不影响当前的播放。也就实现了边下载边播放。

HLS和MPEG DASH 协议

HLS

- apple公司
- m3u8 作为索引文件，ts作为视频资源
- 视频切片
- exp：优酷



DASH

- Dynamic Adaptive Streaming over HTTP
- 基于MSE构建

- 视频内容切片、每个切片都有不同的码率
- DASH Client可以根据网络的情况选择一个码率进行播放，支持在不同码率之间无缝切换
- 推荐视频格式 Fragmented MP4
- 跨平台兼容性比较好
- exp: Youtube，B站

参考资料

https://blog.csdn.net/weixin_40117614/article/details/93018940

https://github.com/marhovey/crypto-worker

https://developer.mozilla.org/zh-CN/docs/Web/API/Media_Source_Extensions_API

https://www.jianshu.com/p/1bfe4470349b

https://juejin.cn/post/6844903880774385671

https://zhuanlan.zhihu.com/p/358216055