文章编号:1671-4598(2019)07-0169-05 DOI:10.16526/j.cnki.11—4762/tp.2019.07.037 中图分类号:TP302.1 文献标识码:A

基于 HLS 协议视频监控加密系统优化设计与实现

王 林,石 焘
(西安理工大学 自动化与信息工程学院,西安 710048)

摘要:为了保证监控视频的安全,并且满足监控视频能够在全平台浏览器无插件播放,设计了一种基于 HLS 协议的视频监控加密系统;服务端将摄像头中视频流实时转码为 HLS 流媒体同时对其加密,客户端借助 HTML5 技术使用浏览器进行解密播放;与现有系统不同,服务端并没有对监控视频所有数据都进行加密,而是采用 SAMPLE-AES 加密方法,对视频中关键数据进行加密,从而减少加密周期,进一步降低解密过程的复杂度;实验结果表明,在采用 SAMPLE-AES 加密方法后,加密的监控视频在全平台的浏览器都能够进行无插件播放,并且在保证安全性的同时客户端在播放视频时 CPU 及内存消耗明显减少;这满足了各个平台客户端使用更小的开销播放加密监控视频的需求,能够有效提升用户体验。

关键词:HLS 协议;监控视频;视频加密;SAMPLE-AES;HTML5

Optimization Design and Implementation of Video Surveillance Encryption System Based on HLS Protocol

Wang Lin, Shi Tao
(School of Automation and Information Engineering, Xi'an University of Technology, Xi'an 710048, China)

Abstract: In order to ensure the security of surveillance video, and meet the surveillance video can be played in the full-platform browser without plug-ins, a video surveillance encryption system based on HTTP live streaming protocol (HLS) is designed. The server transcodes the surveillance video stream in the camera into HLS streaming media in real time and encrypts it. The client uses the HTML5 technology to decrypt and play encrypted surveillance video using the browser. Different from the existing system, the server does not encrypt all the data of the surveillance video, but uses the SAMPLE-AES encryption method to encrypt the key data in the video, thereby reducing the encryption period and reducing the complexity of the decryption process. The experimental results show that after using the SAMPLE-AES encryption method, the encrypted surveillance video can be played without plug-ins in the browser of the whole platform, and the CPU and memory consumption of the client playing the video while ensuring security is significantly reduced. This satisfies the need for each platform client to play encrypted surveillance video with less overhead, which effectively improves the user experience.

Keywords: HLS protocol; surveillance video; video encryption; SAMPLE-AES; HTML5

0 引言

如今视频监控技术取得了很大的进步,用户无论身在何处,只要能够连接网络就能够随时访问监控摄像头。由于网络的开放性,视频监控在给人们提供信息的同时,视频监控的安全问题也逐渐显现,日益威胁公众隐私和治安管理[1]。显然监控视频信息在传输过程中,存在外泄的可能,同时信息也可能被篡改而导致不完整[2]。因此,加强对监控视频信息安全的研究,拥有一个既能保证用户使用便利又能保障信息安全的视频监控加密系统,对公众而言具有重要的意义。

传统的视频监控系统普遍使用实时传输协议(real-time transport protocol, RTP)传输监控视频,并配合使用实时流传输协议(real-time streaming protocol, RTSP)对视频进行控制。使用这些协议的流媒体系统,服务端需要专门的流媒体服务器支持,实现起来较为复杂[3]。另外支持 RTP/RTSP 协议的客户端所需要的软硬件资源较多,加密的视频在浏览器端播放需要额外安装 Flash 插件,并且 Flash 这种第三方插件安全性得不到保障[4]。随着 HTML5 标准的兴起,Web 浏览器在各方面都有了很大的提升,其原生具有跨平台的特性[5]。使用现代 Web 浏览器,借助在移动端有良好支持的 HTTP Live Streaming (HLS) 流媒体传输协议[6],可以同时在移动端及个人计算机(personal computer, PC)端在不使用第三方插件的情况下直接播放加密监控视频。但是这些现有的系统在监控视频的安全防护方面还存在不足,这些系统要么仅仅采用权限控制的方式保护视频,并没有对视频本身加密;要么使用 HLS 协议中 AES-128 加密方法对监控视频中所有数据都进行了加密。然而这些视频数据中有一些非必要的内容是不需要加密的,只要将关键数据保护好就能保障视频的安全。更多的加密量需要更多的加密周期,过多的加密周期进而会产

收稿日期:2019-01-16; **修回日期**:2019-01-29。
基金项目:陕西省科技计划重点项目(2017ZDCXL-GY-05-03)。
作者简介:王 林(1963-),男,江苏东台人,博士,教授,主要从事复杂网络、信息安全、智能视频监控方向的研究。

生不必要的资源消耗，特别是对资源有限的客户端来说，在解密视频过程中不可避免地占用过多的 CPU 及内存资源，影响用户体验。

针对以上问题的不足，本文对现有的视频监控加密系统进行优化设计，将 HLS 协议中的 SAMPLE－AES 加密方法应用到系统中，对视频中少量的关键数据进行加密，对大量的非关键数据不加密。在保证安全性的同时，客户端播放视频时能够有效减少 CPU 及内存资源的占用，降低客户端资源的消耗。并借助 HTML5 及媒体源扩展（media source extensions，MSE）技术，实现加密监控视频在 PC 端、iOS 移动端、Android 移动端的浏览器中无插件地播放的目的。

1 视频监控加密系统整体架构

本视频监控加密系统基于 HLS 协议，能够将网络监控摄像头中的视频流实时编码为 HLS 流媒体，同时对视频中的关键数据进行选择性加密保护，经过身份验证后客户端能够在全平台浏览器无插件地播放监控视频。该系统分为监控视频源、服务端和客户端，其中服务端包括编码加密模块、内容分发模块、身份认证模块。视频监控加密系统整体架构如图 1 所示。

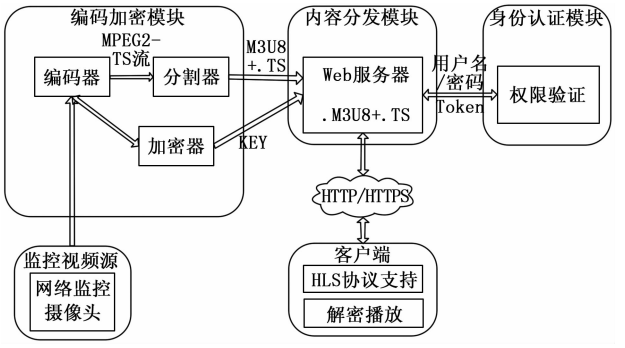

图 1 视频监控加密系统整体架构

如图 1 可知，监控视频源为网络监控摄像头，采集视频数据并向服务端提供视频流。服务端通过摄像头提供的视频地址获得视频流，编码器将视频流进行指定格式的编码，同时调用加密器对视频流进行 SAMPLE－AES 加密，加密后封装为 MPEG－2 传输流（transport stream，TS）。然后分割器将 TS 流切分为 TS 切片文件并产生索引列表文件，并提供给内容分发模块。客户端想要播放视频，首先通过统一资源定位符（uniform resource locator，URL）地址向内容分发模块发送请求，并经过身份认证模块的验证。认证通过后获得所需文件，进一步解密视频，并转换为浏览器指定的格式进行播放。

2 系统中各模块的详细设计

2.1 监控视频源

监控视频源由 RTSP 网络监控摄像头提供，网络监控摄像头由模拟摄像模块和网络编码模块组成。在网络监控摄像头内部，模拟摄像模块采集模拟媒体信号，经过编码模块变换成数字媒体流信号，然后将其编码后封装到 RTP 包中，并通过有线或无线以 RTSP 视频流的形式输出。

2.2 编码加密模块

编码加密模块包括编码器、加密器和分割器。

2.2.1 编码器

编码器负责将从监控视频源接入的 RTSP 视频流实时编码并重新封装。首先读取来自网络摄像头的 RTSP 视频流，解析视频流中 RTP 数据包并得到原始基本流（elementary stream，ES），将视频 ES 基本流进行 H.264 编码，音频 ES 基本流进行 AAC 编码。在编码后的 ES 基本流的包头中分别添加显示时间标记（presentation time stamp，PTS）、解码时间标记（decoding time stamp，DTS）并打包成分组基本流（packetized elementary streams，PES）。最后将节目专用信息（program specific information，PSI）加在 PES 流上并经过复用器将视频数据封装到 MPEG－2 TS 传输流中。具体封装过程如图 2 所示。

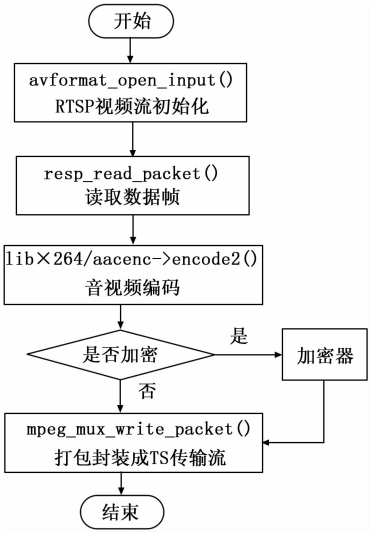

图 2 封装成 TS 流具体过程

如图 2 所示，在将进行了 H.264 编码、AAC 编码之后的音视频 ES 流打包之前要做一个判断，如果监控视频不需要加密，则进行下一步的打包封装操作；如果需要加密，则调用加密器，对音视频进行加密后再进行打包封装操作。

2.2.2 加密器

加密器使用 SAMPLE－AES 样本加密方法，先对音视频 ES 基本流进行加密再对其打包封装。首先生成 128 位的密钥文件用于加密并将 URL 地址发送给分割器，同时选择需要加密的数据块为下一步加密做准备。H.264 视频加密块是指定类型的网络提取层（network abstraction layer，NAL）单元，音频加密块是音频帧。每个需要加密的 NAL 单元或音频帧都包含整数个 16 字节块。然后使用 128 位密钥文件采用 AES 加密算法[7]的密码块链接（cipher block chaining，CBC）模式对 16 字节数据块进行加密，无需填充，在每个 NAL 单元或音频帧的开始处初始化向量（initialization vector，IV）被重置为其原始值。下一步的封装

及分割不会对加密的音视频产生影响。

对 H.264 视频具体加密过程如图 3 所示，只加密类型为 1 和 5 的 NAL 单元，其他类型 NAL 单元不加密。

以下是加密的 NAL 单元的代码格式：

```
Encrypted_nal_unit(){
nal_unit_type_byte    // 1 个字节
unencrypted_leader    // 31 个字节
while(bytes_remaining()＞ 0){
if(bytes_remaining()＞ 16){
encrypted_block // 16 个字节
}
unencrypted_block    //1－144 字节
}
}
```

每个加密的 NAL 单元都包含防止二义性的前缀，即包含 nal _ unit _ type 值的字节和后面的 31 个字节，前缀不加密。未加密字节后面是需要加密的数据块。任何长度不超过 16 个字节的数据块都不需要加密，因此长度为 48 字节或更少的 NAL 单元是完全未加密的。

如图 3，使用 10%跳过的加密方式对 NAL 单元数据块加密。即先加密 16 个字节的数据块，然后跳过剩下的 90%，后面最多有 9 个 16 字节即 144 字节的数据块不加密，第 10 个 16 字节数据块继续进行加密，接下来加密形式以此类推。其他类型为 1 和 5 的 NAL 单元只要长度大于 48 字节，在被加密时，要在整个 NAL 单元上再次加上前缀。

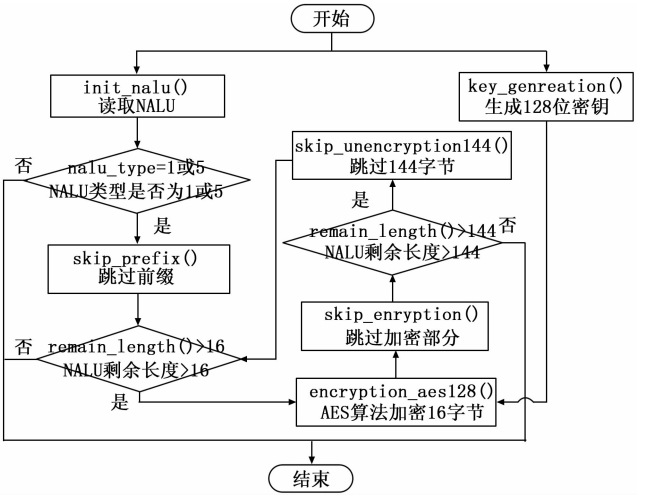

图 3　H.264 样本加密流程图

客户端在解密 H.264 视频时，首先需要识别类型为 1 和 5 同时长度大于 48 字节的 NAL 单元，然后除去防止二义性的前缀。最后定位到 NAL 单元的加密部分，并解密该部分的数据。

AAC 音频流的加密数据块是包含音频数据传输流（audio data transport，ADTS）头的音频帧，加密格式如下所示：

```
Encrypted_AAC_Frame(){
ADTS_Header        // 7 或 9 个字节
unencrypted_leader // 16 个字节
while(bytes_remaining()＞ ＝ 16){
encrypted_block // 16 个字节
}
unencrypted_trailer // 0－15 个字节
}
```

ADTS 头可以是 7 或 9 个字节长，加上后面音频帧的前 16 个字节，这些数据不加密，随后的连续数据部分被加密。加密部分的大小必须是 16 字节的整数倍，并且可能为零。

2.2.3　分割器

分割器负责将编码后的 MPEG－2 TS 流切分成一系列连续且播放时间相等的很小的 TS 切片文件，然后将其发送到内容分发模块进行存储。这些切片文件的大小由用户提前设定。如图 4 中所示，切分过程中要将 DTS 差值与用户设定的时间长度进行比较，只有达到用户设定的长度时才生成当前切片文件。在对 TS 传输流进行具体分割时要注意每个切片文件中都必须含有一个节目关联表（program association table，PAT）和一个节目映射表（program map table，PMT），同时还要保证必须含有至少一个关键帧和序列头等信息，从而完成解码器的初始化。

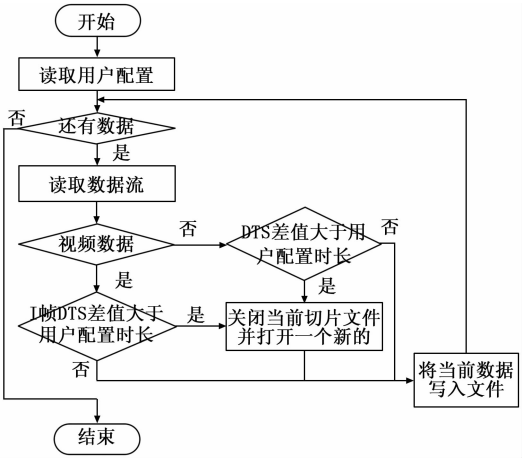

图 4　TS 切片文件生成流程图

切分过程中要根据媒体流中每帧图像的 DTS 值来计算 DTS 差值。DTS 差值算法流程如图 5 所示。每生成一个新的 TS 切片文件，都要记录第一帧图像的时间戳并记作 LastTimestampInStream，将该时间戳作为该切片文件的起始时间。后续的图像都要根据其当前时间戳 CurrentTimestampIntream 减去起始时间 LastTimestampInStream，从而计算出当前 TS 切片文件的时长。当 DTS 差值达到用户提前设定的 TS 切片文件时长时，便关闭当前 TS 切片文件，准备进行下一个 TS 切片文件的写入。

在对 TS 传输流进行分割的同时，分割器还要创建一个含有这些 TS 切片文件 URL 地址的 M3U8 索引列表文件，同样发送到内容分发模块存储。每当分割器生成一个新的 TS 切片文件时，这个 M3U8 索引列表文件将会更新，新生成的 TS 切片的 URL 地址加入到索引列表文件末尾，同时

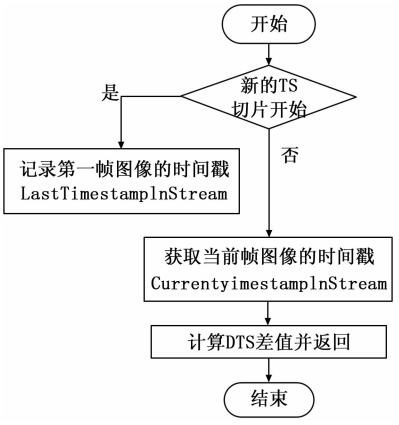

图 5 DTS 差值算法流程图

时间最久的处于索引列表开头部分的切片文件 URL 地址被移除。如果视频数据被加密，还需要添加 # EXT－X－KEY：METHOD＝SAMPLE－AES，URI＝”KEY 文件的地址”标签，表示视频已经使用 SAMPLE－AES 加密方法加密，并指出相应的加密密钥文件的 URL 地址。

2.3 内容分发模块

内容分发模块是一个标准的 Web 服务器，负责将加密并分割后的 TS 切片文件、密钥文件及 M3U8 索引列表文件通过 HTTP 传输协议发送到请求的客户端。为了进一步增加传输过程中的安全性，本系统中索引列表文件及密钥文件采用 HTTPS 协议进行传输层安全性协议（transport layer security，SSL）加密后发送给客户端。要支持 HLS 协议规定的媒体类型文件的 GET 请求，只需要在 Web 服务器的媒体多用途互联网邮件扩展（multipurpose internet mail extensions，MIME）类型中添加如表 1 所示配置[8]。

表 1 媒体 MIME 类型配置

MIME 类型	文件扩展名
Application/x－mpegURL 或 vnd. apple. mpegURL	. m3u8
Video/MP2T	. ts

2.4 身份认证模块

对于被加密的监控视频，由于其 M3U8 索引列表文件中直接列出了 TS 切片及密钥文件的 URL 地址，只要获得了索引列表文件，直接就能获得密钥文件并解密视频。因此本系统对索引列表文件及加密密钥文件同时设置了访问权限验证，确保了由加密、传输以及访问权限控制的整个过程的安全。

本模块采用的是基于 Token 的身份认证机制[9]，相对于传统的 Cookie/Session 机制，Token 认证扩展性更强、更安全，有支持跨域访问、无连接状态、去耦合化和更好的性能体验等优势。Token 在服务端产生，客户端使用用户名和密码向身份认证模块请求认证，认证成功后服务端为其颁发一个独有的 Token 凭证。客户端可以在每次请求密钥文件的时候在 HTTP 请求的头部分附加 Token 证明自己的合法身份，Token 认证过程如图 6 所示。为了保证用户名、密码在验证过程以及 Token 信息在传输过程的安全，采用 HTTPS 协议对其进行 SSL 加密后传输。

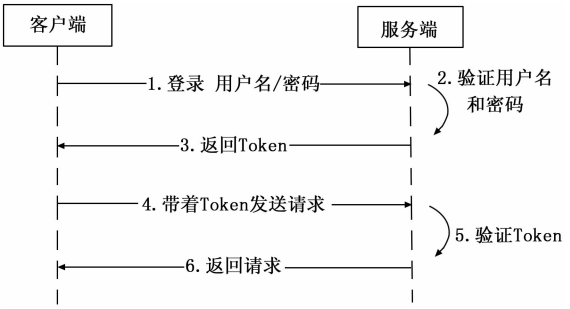

图 6 Token 认证过程图

2.5 客户端

客户端包括 PC 端、iOS 移动端、Android 移动端的 Web 浏览器，用户在浏览器上首先经过身份验证，认证通过后通过指定的 URL 地址，基于 HTTP 请求来获取和下载索引列表文件，并通过索引列表进一步获得 TS 切片文件和密钥文件。接下来浏览器用密钥文件解密 TS 切片并将解密后的 TS 切片直接便捷地使用 HTML5 中新增的＜video＞和＜audio＞标签来播放。

目前现代浏览器都支持 MTHL5 技术和 MSE 技术，但是 PC 端一些主流浏览器（如谷歌浏览器、火狐浏览器）还不支持 HLS 协议中的 TS 传输流。为了使这些主流浏览器也能够无插件播放视频，本系统在客户端请求的网页中嵌入了 hls. js 开源库[10]。

开源库 hls. js 基于 HTML5 和 MSE 技术，原理是通过调用 MSE API 来使用 JavaScript 不使用任何插件动态地将 MPEG－2 TS 传输流转换为 MP4 片段并提供给＜video＞和＜audio＞标签，同样满足在各个平台无插件播放的目的。

3 系统实现及结果分析

3.1 系统实现及播放效果

本系统的开发环境为 CentOS Linux 7 操作系统，网络摄像头型号为海康威视 DS－2CD1221D－I3，根据摄像头的型号得到监控视频取流的 RTSP 地址，本系统采用默认的用户名密码，地址为：rtsp：//admin：12345@172. 6. 22. 234：554/ Streaming/Channels/101？ transportmode ＝ unicast。然后将这个 RTSP 视频流地址提供给服务端编码加密模块中的编码器。

编码加密模块基于 FFmpeg 开源编解码框架，首先通过调用 AVInputFormat 结构体中的 ff _ rtsp _ demuxer 函数解析 RTSP 视频流；然后调用 AVCodec 结构体中的 ff _ libx264 _ encoder 函数对视频进行编码；最后调用 AVOutputFormat 结构体中的 ff _ mpegts _ muxer 函数和 ff _ stream _ segment _ muxer 函数将依次对视频进行封装和切分。需要对其中 ff _ libx264 _ encoder 函数中的 X264 _ frame （） 函数进行修改，在 encode _ nals （） 函数后添加拥有 SAMPLE－AES 加密功能的 nal _ sample _ encrypt （） 函数，用于对编码后 NAL 单元进行加密。

内容分发模块采用内存占用小、性能稳定的 Nginx 作为 Web 服务器。身份认证模块没有单独另外搭建一个认证服务器，而是通过将 lua 语言嵌入到高性能的 Nginx 服务器中，使 Nginx 可以高并发、非堵塞地处理 Token 认证请求，并利用 redis 键值数据库的超时机制设置 Token 时效性来控制 Token 的有效性。由于 OpenResty 是一个集成了 Nginx 与 lua 的高性能 Web 平台[11]，因此本系统直接安装 OpenResty 与 redis，共同实现内容分发及 Token 认证的功能。安装后需要编写 lua 脚本实现 Token 验证，包含的主要脚本有：auth _ req _ headers. lua，请求头校验脚本，失败直接中断请求；auth _ token. lua，Token 处理脚本；handle _ cors. lua，请求跨域脚本；handle _ request _ provision. lua，请求 handle 入口脚本；redis _ mcredis. lua，redis 操作工具的封装脚本；tool _ dns _ server. lua，域名解析，获取域名对应 IP，并设置缓存的脚本。

系统搭建好之后，观察播放效果，如图 7 所示，PC 端的谷歌浏览器，iPad 端的 Safari 浏览器、Android 端谷歌浏览器及微信端网页各个平台都能满足使用 HTML5 技术直接无插件地播放加密监控视频。

图 7　各个平台播放效果图

3.2　安全性分析

NAL 单元中最重要的单元类型是 1 和 5，包含了所有的视频数据。类型为 5 的 NAL 单元负载中包含的是立即刷新图像（instantaneous decoding refresh，IDR）的图像片段，类型为 1 的 NAL 单元负载包含的是非 IDR 帧的图像片段，解码器在收到 IDR 数据单元后会立即刷新所有图像，并作为之后的所有数据的解码参照。因此对这些最重要的包含了图像片段的 NAL 单元类型进行选择性加密，即使其他类型不加密，非授权用户也无法正确解码视频。并且加密过程采用的是 AES 加密算法，AES 加密算法为新一代数据加密标准，能够同时满足强安全性、高效率、高性能、易用和灵活的特点[7]。AES 加密算法是目前可获得的最安全的对称加密算法，密钥长度达到 128 位就能达到保护机密信息的标准。另外 NAL 单元中相对固定的头部信息没有加密，可以有效防止明文攻击。在没有密钥的情况下播放加密的监控视频效果如图 8 所示。

同时，本系统采用高度安全的 HTTPS 协议对索引列表文件及密钥文件进行传输，给攻击者增加了巨大的难度，根本无法获得密钥信息。客户端采用了安全性更高的基于 Token 的身份认证机制有效拦截了非授权用户的播放。

图 8　加密后播放效果图

3.3　实验结果对比分析

将本系统与现有使用 AES－128 加密方法的其他监控加密系统进行对比，检测客户端播放加密视频过程中的 CPU 占用率及内存使用率。采用同一个监控视频源，相同播放平台，都是借助 HTML5 进行播放，不同的是监控视频在不同加密系统中使用不同加密方法加密。

本次实验的播放平台选用内存为 16 GB，处理器为 Inter i5－4590 CPU@3.30 GHz x 4 的 PC 端的谷歌浏览器。客户端在播放视频过程中 CPU 使用率对比结果如图 9 所示，客户端内存占用率对比结果如图 10 所示。

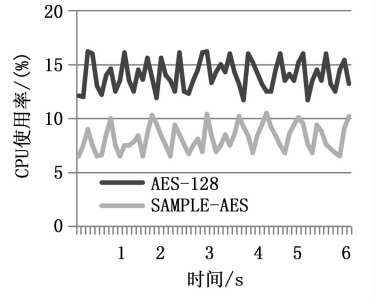

图 9　客户端 CPU 消耗对比

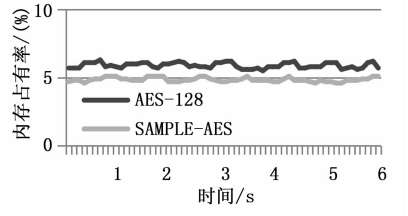

图 10　客户端内存占用率对比

由图 10 及图 11 实验结果可知，本系统由于采用 SAMPLE－AES 加密方法，没有对所有视频数据都进行加密，而是对监控视频中包含视频片段的 NAL 单元进行选择性加密，从而有效减少加密周期，进一步降低了客户端解密过程的复杂度，减少了在 CPU 占用及内存使用中而产生的不必要的消耗，用户体验得到有效提升。

4　总结

本文对现有的视频监控加密系统进行了优化设计，将 HLS 协议中的 SAMPLE－AES 加密方法应用在系统中，选择性地加密视频中的关键数据。与现有系统相比，在保证安全性的同时播放加密视频的客户端 CPU 及内存消耗明显

（下转第 179 页）