Secure Real-Time Streaming Protocol (RTSP) for Hierarchical Proxy Caching

Yeung Siu Fung, John C. S. Lui, and David K. Y. Yau

(Corresponding author: John C. S. Lui)

Department of Computer Science & Engineering The Chinese University of Hong Kong
Shatin, Hong Kong (Email: cslui@cse.cuhk.edu.hk)

(Received Aug. 26, 2006; revised Oct. 6, 2006; and accepted May 1, 2007)

Abstract

Proxies are commonly used to cache objects, especially multimedia objects, so that clients can enjoy better quality-of-service (QoS) guarantees such as smaller startup latency and lower loss rate. But the use of multimedia proxies increases the risk that data are exposed to unauthorized access by intruders. In this paper, we propose an enhancement of the Internet IETF's Real-time Streaming Protocol (RTSP) which employs a notion of "asymmetric reversible parametric sequence" (ARPS) to provide the following security properties: (i) data confidentiality during transmission, (ii) end-to-end data confidentiality, (iii) data confidentiality against proxy intruders, and (iv) data confidentiality against member collusion. We present the Secure Multimedia Library (SML) which is based on ARPS and then realize these security features on a production video streaming server: Apple's Darwin Streaming Server. Our framework guarantees the system resilience against attacks is provably strong given the standard computability assumptions. To reduce the computation demand on the receiving client, our scheme only requires the client to perform a "single decryption operation" to recover the original data even though the data packets have been encrypted by multiple proxies along the delivery path. To tradeoff between degree of confidentiality and computational overhead, we also propose the use of a set of "encryption configuration parameters" (ECP) to trade off proxy encryption throughput against the presentation quality of audio/video obtained by unauthorized parties. Our implementation prototype shows that one can simultaneously achieve high encryption throughput and extremely low audio/video quality (in terms of audio fidelity, and peak signal-to-noise ratio and visual quality of decoded video frames) for unauthorized access.

Keywords: Security, asymmetric parametric sequence functions, multi-key RSA, video proxy, real-time streaming protocol

1 Introduction

Recently, the transition from modem network to broadband network has proceeded at an amazing pace. Most urban areas are now within the broadband coverage as the availability of broadband services increase continuously. At the same time, subscription fees of broadband services have dropped to a very comparable and reasonable rate that modem connections begin to fade out the market. The major broadband technologies such as Asymmetric Digital Subscriber Line (ADSL), Very-High-Data-Rate DSL (VDSL) and Cable-Modem provide Internet connections with bandwidth sufficient for high quality digital entertainment applications such as MPEG-4 video streaming, which offers a near DVD quality at a bit-rate of around 1 Mb/sec. However, the increasing capability at the client side imposes challenges at the server side and the Internet Infrastructure.

For a streaming server equipped with a gigabit network interface, the maximum number of concurrent clients that can be served is limited at around the order of hundreds. However, compared to conventional or non-multimedia applications, the total traffic generated by these multimedia streams is extremely large and will be very likely to congest the Internet. Deploying multimedia proxies is a very effective means to support a large number of concurrent clients while cutting down Internet traffics. However, some multimedia streaming applications require data confidentiality, which prohibit the caching of plain-text data. Moreover, these applications would like to use end-to-end encryption technique on the multimedia data that makes caching of the data impossible unless the proxies have knowledge on the decryption keys, which is undesirable for security reasons.

To protect data from intruders, end-to-end encryption is used so that one cannot reveal any meaningful data by eavesdropping on any intermediate channels between the server and the clients. Since data is encrypted with a specific session key where only the dedicated client has the proper decryption key for decryption, a proxy which caches the encrypted data cannot provide the QoS guarantees to other clients since these clients do not possess the same session decryption key. To enable secure proxy caching while not sacrificing any data confidentiality nor encryption security, the multimedia data should be encrypted using an encryption scheme which satisfies all of the following requirements:

- **Data confidentiality during transmission** - because intruders may eavesdrop on the transmission paths, therefore, multimedia data need to be transmitted across the network in encrypted form.
- **Data confidentiality against member collusion** - to avoid compromise of the system by a compromised client, each decryption keys should only be valid for a specific session. And the knowledge of the decryption key for any particular session should not lead to any knowledge of other decryption keys.
- **Data confidentiality against proxy intruders** - for the purpose of data proximity to different clients, proxies are intended to distribute over the Internet on different local domains. The system can be compromised by a compromised proxy. However, it is impractical to control all proxies by the content owner. Therefore, the proxies must be assumed unauthentic and the data confidentiality features should not rely on the security level of these proxies. A proxy should not perform any decryption operation and it will not have any knowledge of the decryption keys.
- **Data encryption security** - although satisfying the above requirements, intruders may still gain access to an encrypted copy of the multimedia data, e.g. by eavesdrop on the communication channels or break into the proxies. Hence, the encryption algorithm used must be proven to be secure, or at least, computational infeasible to break.
- **Encryption scalability** - since the proxy infrastructure on the Internet are hierarchical in nature, a proxy may fetch multimedia data from another proxy. This hierarchy should be transparent to the end-clients. And for scalability reason, the number of proxy layers should not affect the computational complexity of the decryption operations at the end-clients.

In our previous paper [12] , we proposed a secure proxy architecture which is based on the concept of asymmetric reversible parametric sequence (ARPS) that can achieve the goals listed above. In this paper, we present an implementation of this proxy architecture as an extension of the Internet IETF's Real-time Streaming Protocol (RTSP). We also present an implementation of the proposed security features as the Secure Multimedia Library (SML) which is based on ARPS and then realize these security features on a production video streaming server: Apple's Darwin Streaming Server. Our framework guarantees the system resilience against attacks is provably strong given the standard computability assumptions. To reduce the computation demand on the receiving clients, our scheme only requires the client to perform a "single decryption operation" to retrieve the original data even though the data packets have been encrypted by multiple proxies along the delivery path. To tradeoff the degree of confidentiality for computational overhead, we also propose the use of a set of "encryption configuration parameters" (ECP) to trade off proxy encryption throughput against the presentation quality of audio/video obtained by unauthorized parties.

The outline of this paper is as follows. In Section 2, we present some related works in the field of proxy caching of multimedia objects and secure multimedia proxy caching, and we give a brief review on the secure multimedia proxy architecture we proposed in the paper [12]. In Section 3, we present an overview of the RTSP and RTP protocols. In Section 4, we present our security extension of the RTSP protocol. In Section 5, we present the experiments that were carried out on our prototype system. Conclusion is given in 6.

2 Background and Related Work

Recent research on video proxies has mainly focused on caching and multimedia object replacement algorithms. In [11], authors present how *prefix caching* at a proxy can reduce large start-up delay, low throughput and packet loss. In [5] , Guo *et al.* propose the use of a prefix-caching proxy in conjunction with a periodic broadcasting technique to improve system scalability. Cruber *et al* [4] focus on implementation and protocol issues and show how to realize proxy prefix caching by using the Real-Time Streaming Protocol (RTSP). Rejaie *et al.* [10] present a fine-grained replacement algorithm for a multimedia proxy, which targets layered-encoded streams. The fine-grained replacement algorithm enables the proxy to perform more effective quality adaptation while the quality of the delivered stream can be maximized. Kangasharju *et al.* [7] present a caching model of layered-encoded multimedia streams, and propose utility heuristics whose performance are evaluated through their caching model.

There are only a small set of papers emphasize on security issues in a video proxy. Griwodz *et al.* [3] propose an approach in which the proxy stores the major part of the video streams which are intentionally corrupted. The proxy can distribute the corrupted part via multicast transmission, while the origin server will supply the part for data reconstruction in an unicast manner. However, because the original server must perform data encryption for each client, this is not a scalable solution. Tosun and Feng [15] propose a much more scalable approach based on a lightweight encryption algorithm for multimedia streams. When a client makes a request, the proxy will decrypt the locally stored encrypted data and encrypt it again using the client's encryption key. The major drawback with their approach is that the use of light-weight encryption offers no proven resilience against attacks on data confidentiality, and the system can be compromised by a compromised proxy. Furthermore, the need for decryption operations at the proxy results in higher computational overhead. Shi and Bhargava [13] present an MPEG video encryption algorithm called VEA such that one can encrypt a video stream multiple times (each with, say, a client-specific key) and one can still decrypt the video only by a single operation using a composite decryption key. However, VEA is not resilient against plain-text attack. Therefore, determined adversaries can obtain the VEA secret key with feasible efforts.

2.1 Asymmetric Reversible Parametric Sequence

In [12], we used the notion of Asymmetric Reversible Parametric Sequence (ARPS)[8] to support a multimedia streaming system and fulfills all of the above requirements. In the proposed proxy architecture, a proxy only stores encrypted multimedia objects, and the proxy does not hold the proper key required to decrypt the encrypted objects. However, a proxy needs to perform re-encryption on the multimedia objects before delivers it to a client or another proxy. As a property of the ARPS, no matter how many proxies are along the streaming path, only a single decryption operation is required on the client side. Since different keys are used to re-encrypt the multimedia objects, a client cannot decrypts the encrypted content received by another client.

In this paper, we provide the software tools required to implement a secure proxy system supported by ARPS. To demonstrate the usage of the software tools and the feasibility of the proposed proxy architecture, we realize all of the security features into a production multimedia streaming system. We developed a C language API that provides the tools required to realize all of the ARPS operations. The API provides low level functions such as session key generation and encryption/decryption on block data. It also provides high level functions such as network connections and encryption/decryption on TCP streams or UDP packets. Using this API, one can easily develop a secure proxy system supported by ARPS, or easily add ARPS features into an existing proxy system. We demonstrate this by extending a production RTSP streaming system with the ARPS functions provided by the API. The extended system is compatible with standard RTSP client software on un-protected multimedia objects, and supports both client-server and client-proxy-server architectures simultaneously.

3 RTSP and RTP Overview

The Real-Time Streaming Protocol (RTSP) is an application-level protocol for control over the delivery of data with real-time properties. RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as multimedia data. The stream controlled by RTSP may use Real-Time Transport Protocol (RTP), but RTSP and the underlying protocol used to carry the media stream will not have any dependence over each other. RTSP is a text-based protocol that makes it easily to be extended. Two ways to extend RTSP are:

- Extending existing methods with new parameters. Recipient would ignore these new parameters if they do not recognize them.
- Adding new methods. If the recipient does not understand the request, it would responds with error code 501.

Some RTSP methods are only recommended (or optional) in practical implementation, a fully functional RTSP application may only implement some of the RTSP methods. Three of the required RTSP methods are SETUP, PLAY and TEARDOWN. These three methods are necessary and sufficient for a client from initiation of multimedia data streaming to termination of the streaming session. A client issues a SETUP request to initiate a server to start a new streaming session, and then issues a PLAY request to initiate streaming of multimedia data, and finally issues a TEARDOWN request to close the streaming session. The sequence of requests and responses, and the corresponding RTSP methods involved in a typical RTSP session, where an RTSP proxy is in between the path of the RTSP server and the RTSP client, are as follows:

1) The client sends a SETUP request to the proxy. The request specifying the destination address the client will use to receive the media stream and the absolute URI of the requesting media object.

2) The proxy replaces the destination address with its own one. And then forwards the modified request packet to the appropriate server by observing the absolute URI insides the request.

3) The server replies the proxy with a SETUP response, specifying the source address of the requested stream and a unique identifier used to associate the streaming session. The proxy allocates all resources and sockets required to relay the media stream, and then forwards the respond to the client.

4) The client sends a PLAY request to the proxy to initiate the streaming of media data. The proxy forwards the request to the server.

5) If the server uses RTP to carry the media data, it will start sending RTP packets to the destination address, which is the address replaced by the proxy. The proxy receives the RTP packets and caches them in its local storage, and at the same time relays the packets to the client.

Each RTP data packet will contain an RTP header that includes a sequence number. The initial value of the sequence number is random and unpredictable, which makes known plaint-text attack more difficult. The sequence number then increases by one for each RTP data packet sent. The server continues to send RTP packets until reaching the end of the media, or the client issues a PAUSE, STOP or TEARDOWN request. The server and the proxy will close the streaming session and release all allocated resources and sockets when the client issues a TEARDOWN request.

4 RTSP with ARPS Extension

To realize our secure system, we integrate new ARPS parameters into existing RTSP methods. In this section, we will describe the implementation details of our extended RTSP with secure proxy extensions based on ARPS.

We use a scenario to illustrate. Let an RTSP client request for a media object "sample.mpeg" from the RTSP server at rtspserver.com. The client requests through an RTSP proxy where the media object is not yet cached. Figure 1 illustrates the operations between the RTSP server and the RTSP proxy, as well as the operations between the RTSP proxy and the RTSP client in this scenario. Throughout the remaining of this article, the terms client, proxy and server refer to RTSP client, RTSP proxy and RTSP server respectively.

4.1 Authentication

We assume that the server has a list of user names of its certificated users and proxies. However, the server must have a mean to obtain the client's public key and the proxy's public key with creditability. We use the X.509 authentication service [2] to perform public key authentication. Suppose that all clients, all proxies and the server subscribe to the same certificate authority (CA). The certificate of a particular entity I is denoted by $CA \ll I \gg$, which includes the following fields:

- **Version:** Version number of this X.509 certificate.
- **Serial number:** An unique integer value within the issuing CA.
- **Signature algorithm identifier:** The algorithm used to sign the certificate.
- **Issuer name:** X.500 name of the CA that created and signed this certificate.
- **Period of validity:** The first date and the last date on which the certificate is valid.
- **Subject name:** The user name to whom this certificate refers.
- **Subject's public-key:** The public key of the subject.
- **Issuer unique identifier:** The unique identifier of the issuing CA.
- **Subject unique identifier:** An unique identifier of the subject.
- **Extensions:** A set of one or more extension fields.
- **Signature:** The hash code of the other fields in this certificate that encrypted with the CA's private key.

The client will include its X.509 certificate in the SETUP request using the extended parameter Signature. The client establishes a transport layer connection to the proxy and then sends the following SETUP request:

```
SETUP rtsp://rtspserver.com/sample.mpeg RTSP/1.0
CSeq: 1004
Transport: RTP/AVP;unicast;client_port=4588-4589
Signature: (x.509 certificate of client-1)
```

The proxy modifies the value of the client_port parameter to the port range that it will use to receive media data from the server. The proxy adds its public key certificate using the extended parameter ProxySignature. The proxy then establishes a transport layer connection to the media server at rtspserver.com and sends the following SETUP request:

```
SETUP rtsp://rtspserver.com/sample.mpeg RTSP/1.0
CSeq: 1004
Transport: RTP/AVP;unicast;client_port=4588-4589
Signature: (x.509 certificate of client-1)
ProxySignature: (x.509 certificate of proxy-1)
```

Because the server subscribes to the same CA, thus it must have a copy of the CA's public key. The server retrieves the hash codes by decrypting the **Signature** field of the proxy's certificates and that of the client's certificate using the CA's public key. The server can then verify the integrity of the two certificates, and thus the integrity of the two public keys, using the corresponding hash codes.

4.2 Session Key Distribution

Once the SETUP request is granted, the server will generate an encryption key e_0, a re-encryption key e_i, and a decryption key d_i. The server associates e_0 and the set of encryption configuration parameters with a unique identifier S and keeps a record of the values using S as the index. The keys e_i and d_i will be distributed to the proxy and the client respectively. To ensure security, e_i will be encrypted using the proxy's public key so that only the proxy can retrieve it. The decryption key d_i will be encrypted using the client's public key so that only the client can retrieve it, but any proxy along the communication path cannot decrypt the data. The server then replies a SETUP response to the proxy including these two encrypted values, using the extended parameter SessionKey. The server will also include the index S and the encryption configuration parameters, using the extended parameter ECP.

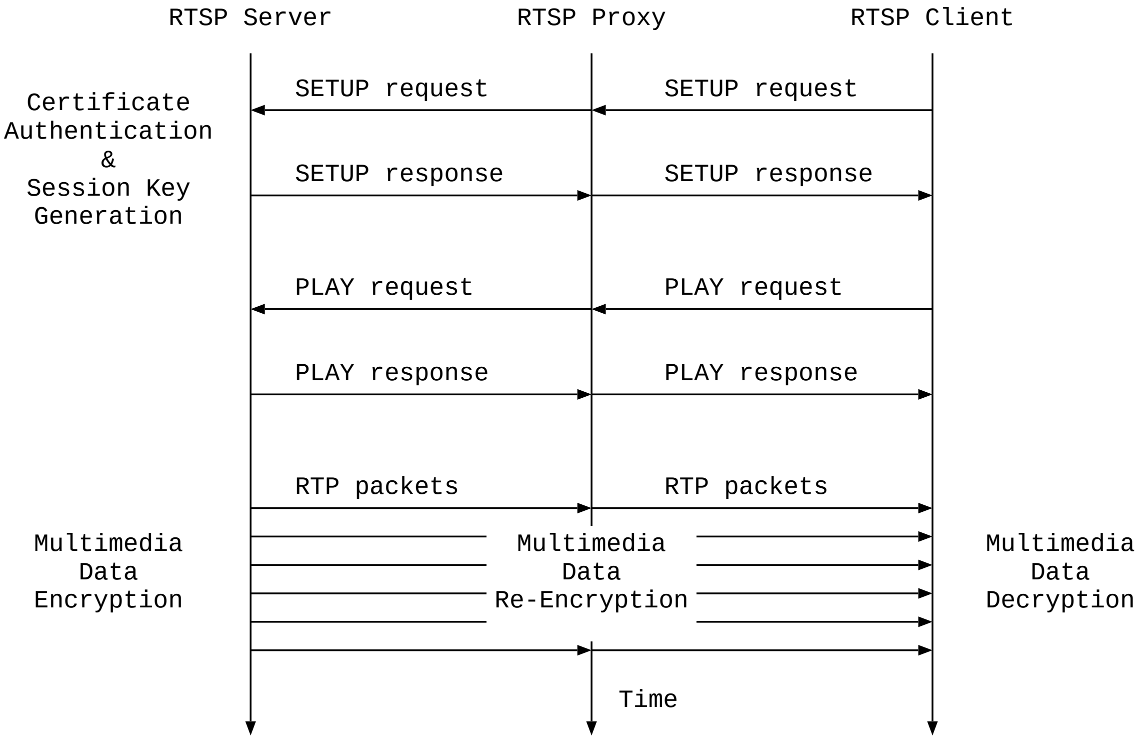


Figure 1: Operations between the RTSP server and the RTSP proxy, as well as the operations between the RTSP proxy and the RTSP client when an RTSP client request an un-cached multimedia object through an RTSP proxy

```
RTSP/1.0 200 OK
CSeq: 1004
Date: 23 Jan 1997 15:35:06 GMT
Server: PhonyServer 1.0
Session: 47112344
Transport: RTP/AVP;unicast;
    client_port=4588-4589;
    server_port=6256-6257
SessionKey: e=uz80989zlkxc01a9zo;
    d=8800x018a83bxc74b5
ECP: S=8012;I=10;P=5;B=1
```

Upon receiving the SETUP response, the proxy extracts the encrypted value of "e" inside the SessionKey parameter and decrypts it using its own private key to retrieve the re-encryption key e_i. The proxy then forwards the SETUP response to the client without any modification. The client extracts the value of the encrypted key "d" and decrypts it using its own private key, which produces the decryption key d_i.

4.3 Media Data Encryption and Decryption

Now, the client may send the PLAY request to initiate the streaming of the media data. The proxy forwards the PLAY request to the server without any modification. The server replies with the PLAY response and starts to stream the RTP packets. According to the encryption configuration parameter E_i and the original sequence number in the RTP header in each RTP packet, the server determines whether the payload of the RTP packet will be encrypted or not, but always leave the header part unencrypted.

The proxy caches the RTP packets directly and associate the cache with the index S given by the server, it observes the sequence number in the unencrypted RTP header to determinate whether to perform re-encryption on the payload or not. The proxy then relays the RTP packets, re-encrypted or not, to the client. The client also observes the sequence number to determinate whether the payload of the RTP packet requires decryption or not.

4.4 Access to Cached Media Object

Referring to figure 2, another client requests for the same media object, “sample.mpeg”, that is now in the proxy’s cache. The client sends a SETUP request to the proxy and the proxy replaces the destination address in the request as in the previous example. However, the proxy examines the request message and knows that the requesting media object is cached. Hence, the proxy retrieves the media object ’s index S and adds it in the SETUP request using the extended parameter Index:

```
SETUP rtsp://rtspserver.com/SAMPLE.MPEG RTSP/1.0
CSeq: 1004
Transport: RTP/AVP;unicast;client_port=4588-4589
Signature: (certificate of client-1)
ProxySignature: (certificate of proxy-1)
Index: 8012
```

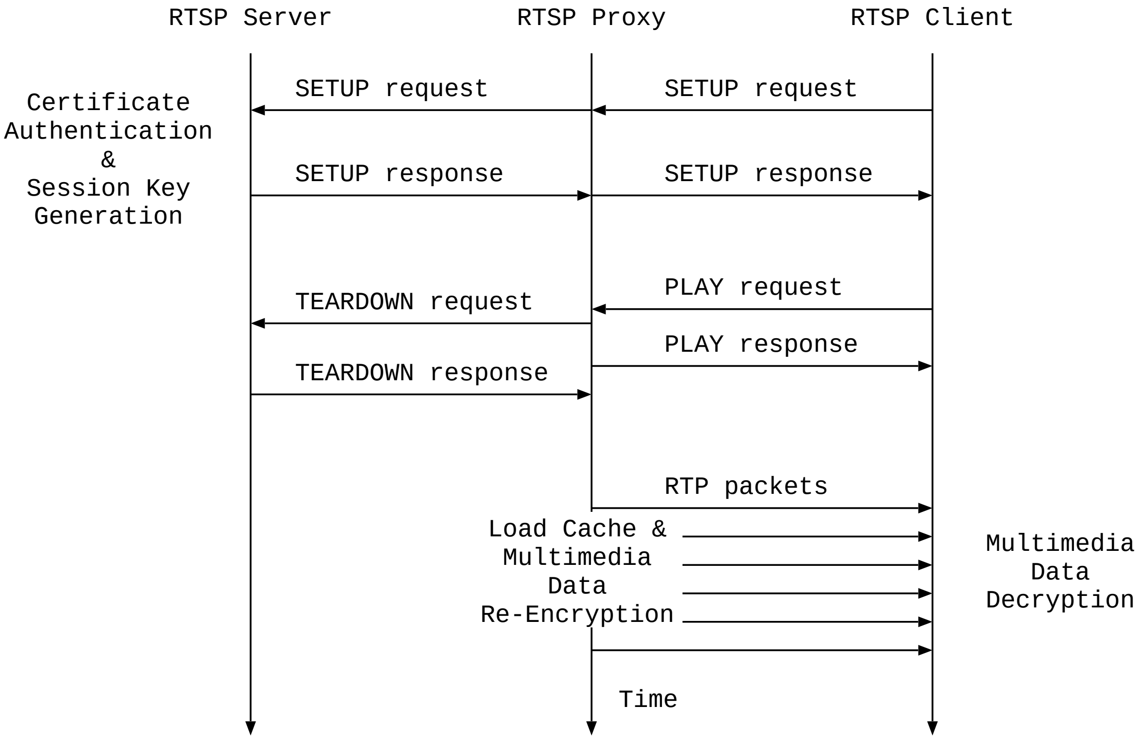


Figure 2: Operations between the server and the RTSP proxy, as well as the operations between the RTSP proxy and the RTSP client when an RTSP client request an cached multimedia object from an RTSP proxy

The server receives the SETUP request and generates a new pair of re-encryption key e_j and decryption key d_j bases on the previous encryption key e_0 associated by the specified index S, and also retrieves the corresponding encryption configuration parameters associated with S. The server then sends back the SETUP response containing e_j encrypted using the proxy's public key, d_j encrypted using the client's public key, and the encryption configuration parameters.

Because the RTP packets are cached at the proxy locally, after receiving the SETUP response successfully, the proxy can send a TEARDOWN request to the server to release all resources allocated by the server and terminate the connection between the proxy and the server. The proxy retrieves the cached RTP packets and uses the new re-encryption key to perform re-encryptions. And the proxy acts as a server to self-control the sending of the cached RTP packets to the client.

4.5 Extension to Hierarchical Multi-Layer Proxy

Here we use a scenario to illustrate how to further extend the system to support multiple layers of hierarchical proxies in-between of the client and the server. Assume that a client requests a media object through a proxy, say proxy $\mathcal{A}$, where the media object has not been cached. However, proxy $\mathcal{A}$ in turn request the media object through another proxy, say proxy $\mathcal{B}$, which has a cached copy of the media object.

Referring to Figure 3, proxy $\mathcal{A}$ sends the SETUP request to proxy $\mathcal{B}$ as it is sending to an origin server, but with the extra parameter HopCount set to 1. When proxy $\mathcal{B}$ receives the SETUP request, it adds its public signature to the SETUP request and increases the HopCount by 1. Because the media object is cached, proxy $\mathcal{B}$ also retrieves the index S associated to the media object, and then sends the following SETUP request to the server:

```
SETUP rtsp://rtspserver.com/sample.mpeg RTSP/1.0
CSeq: 1004
Transport: RTP/AVP;unicast;client_port=4588-4589
Signature: (certificate of client-1)
HopCount: 2
ProxySignature: (certificate of proxy-A)
ProxySignature2: (certificate of proxy-B)
Index: 8012
```

After authenticating all of the identities of the involved parties, the server retrieves the encryption configuration parameters and the encryption key e_0 associated by the index S. The server then generates two re-encryption key e_k and e_{k+1}, and a corresponding decryption key d_k. All session keys need to be protected for their dedicated recipients, that is, encrypts e_k, e_{k+1} and d_k with proxy $\mathcal{B}$'s public key, proxy $\mathcal{A}$'s public key and the client's public key respectively. The server includes the encrypted session keys in the SETUP response, also a HopCount value set to 0, and a new index S_2 used to associate this ARPS that involved two layers of encryption with encryption keys e_0

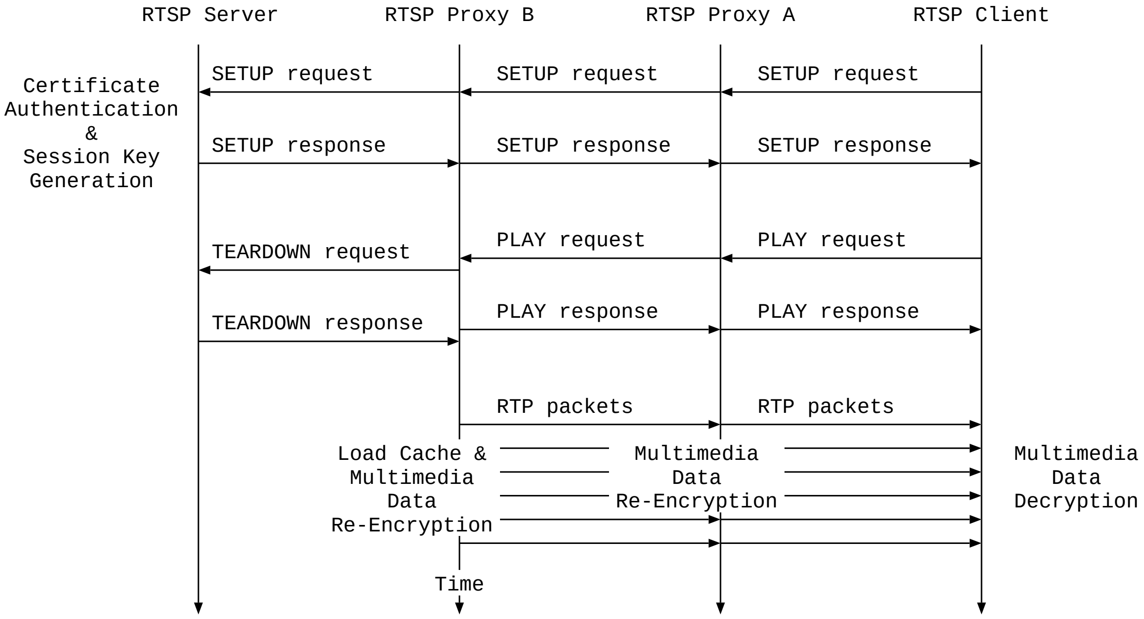


Figure 3: Operations between the RTSP server and the RTSP proxy, as well as the operations between the RTSP proxy and the RTSP client when there are multi-layer of proxy in-between the client and the server

and e_k. The server then replies proxy $\mathcal{B}$ with the following SETUP response:

```
RTSP/1.0 200 OK
CSeq: 1004
Date: 23 Jan 1997 15:35:06 GMT
Server: RTSPServer 1.0
Session: 11146612
Transport: RTP/AVP;unicast;client_port=4588-4589;
    server_port=6256-6257
HopCount: 0
SessionKey: e1= ii77761huajzzzzz73;
    e2=v91110001kzaa9a9xz;
    d=761axzx8gg3bx19283
ECP: S=8013;I=10;P=5;B=1
```

Proxy $\mathcal{B}$ uses its own private key to decrypt e_1 and retrieves the re-encryption key e_k. It then increases the HopCount by 1 before forwards the SETUP response to proxy $\mathcal{A}$. Now proxy $\mathcal{A}$ receives the SETUP response with a HopCount value equals to 2, so it decrypts e_2 with its own private key and retrieves the re-encryption key e_{k+1}. Finally, proxy $\mathcal{B}$ forwards the SETUP response to the client and the client retrieves the decryption key d_k.

Once the PLAY request is received after the SETUP process, proxy $\mathcal{B}$ will retrieve cached RTP packets, performs re-encryption according to the sequence number of each RTP packet and the corresponding encryption configuration parameters, using re-encryption key e_k. And then delivers the RTP packets to proxy $\mathcal{A}$. At the meanwhile, proxy $\mathcal{A}$ caches the RTP packets and associates the cache with index S_2, performs re-encryption using re-encryption key e_{k+1} and deliveries the re-encrypted packets to the client.

5 Experiments on the Darwin Prototype

We have implemented an application-programming interface in C language, which we called Secure Media Library (SML), which provides all of the necessary routines for the ARPS and ECP operations, and provides session key and public key manipulations. We have used SML to implement a prototype of the ARPS-extended RTSP system described in Section 4. Our implementation is based on the Apple's Darwin Streaming Server [1], the client software included in the MPEG4IP project [14] and the proxy reference implementation provided by RealNetworks [9]. We modified the source codes such that the whole system still maintains compatibility to other standard RTSP software, while the ARPS extended client is able to access secure multimedia objects from the ARPS extended server through the ARPS extended proxy.

Our server allows an administrator to set the permission of each individual multimedia object, either be public accessible or only be available to a list of certificated users. To access a secure multimedia object, a user must use the extended client software and specify a file containing a pair of public key and private key. The use of a proxy is configurable at the client side, it can be a standard RTSP proxy, or must be our ARPS extended proxy when accessing a secure multimedia object. Apart from this, the existence of the proxy is totally transparent to the end-user. We carry out the following experiments to quantify the merit and performance of our system.

Experiment 1 (Encryption Throughput Analysis): In this experiment, we consider the effect of varying

Table 1: Effect of E_p and E_i on the encryption throughput ρ (in unit of MBytes/s) and the corresponding average number of clients M when E_b keeps at 1

$\mathbf{E_i \backslash E_p}$	0.257		0.214		0.171		0.120		0.086		0.043	
	ρ	**M**	ρ	**M**	ρ	**M**	ρ	**M**	ρ	**M**	ρ	**M**
1	1.75	9.33	2.23	11.89	2.79	14.88	3.92	20.91	5.73	30.56	10.04	53.55
2	3.43	18.29	4.02	21.44	5.90	31.47	7.59	40.48	8.27	44.11	10.30	54.93
5	7.50	39.99	8.72	46.51	9.82	52.37	10.54	56.21	9.83	52.43	11.18	59.63
10	9.53	50.83	11.71	62.45	11.64	62.08	11.77	62.77	11.39	60.75	11.72	62.51

Table 2: Effect of E_i and E_b on the encryption throughput ρ (in MBytes/s) for (a) $E_p = 0.257$ and (b) $E_p = 0.171$

	encryption throughput ρ (MB/sec)			
$\mathbf{E_b \backslash E_i}$	1	2	5	10
1	1.75	3.43	7.50	9.53
2	1.85	3.75	8.29	11.67
3	1.96	3.81	8.17	11.63

(a) $E_p = 0.257$

	encryption throughput ρ (MB/sec)			
$\mathbf{E_b \backslash E_i}$	1	2	5	10
1	2.79	5.90	9.82	11.64
2	2.32	4.56	11.17	11.69
4	2.87	5.46	11.37	11.73

(b) $E_p = 0.171$

the encryption parameters E_p and E_i on the encryption throughput, which is denoted as ρ (in MBytes/s). We turned off the rate control at the server side and measured the maximum encryption throughput of the server for encrypt and deliver a QuickTime video stream of 54MB in size. Assume that all multimedia objects are encoded into 1.5 Mb/s, the corresponding average number of concurrent clients that a server can support is M, where $M = \rho/(1.5/8)$. The experiment runs on an 800 MHz Pentium-III Linux server with 256 MBytes main memory. We have repeated the experiment three times and the average values are taken. Table 1 illustrates the encryption throughput ρ and the corresponding average number of concurrent clients (M) under different values of E_p and E_i, while E_b keeps at 1.

As we can observe from Table 1, if we encrypt 25.7% of *each* video packet (i.e., $E_i = 1$), the encryption throughput achieved is only around 1.75 MBytes/s, which implies that we can concurrently handle about 9 MPEG-1 streams. On the other hand, if we encrypt one video packet for every 10 packets (i.e., $E_i = 10$) and for each video packet encrypted, we encrypt only 4.3% of its data (i.e., $E_p = 0.043$), then the encryption throughput improves to 11.72 MBytes/s, which implies that we can concurrently support about 62 MPEG-1 streams. In general, the smaller the value of E_p and the higher the value of E_i, the higher the achieved encryption throughput, and the higher the number of concurrent video streams that can be supported.

Table 2 illustrates the effect of varying E_i and E_b under two different encryption percentage parameters E_p. As we can observe, the parameter E_b has little effect on the encryption throughput.

Experiment 2 (Peak Signal-to-Noise Analysis): In this section, we consider the effect on the video quality as we vary the parameters E_i, E_p, and E_b. One way to quantitatively evaluate the video quality is by the peak signal-to-noise ratio. In general, for a frame size of $m \times n$ with a total of l frames and 3 color channels (i.e., red, green, and blue, each represented by a 8-bit number), the peak signal-to-noise ratio (SNR_{peak}) is calculated using the following equation:

$$SNR_{peak} = 10 \times \log_{10} \frac{255^2}{\left(\frac{\sum_{x=1}^{m} \sum_{y=1}^{n} \sum_{z=1}^{l} \sum_{c=1}^{3} (P_1(x,y,z,c) - P_2(x,y,z,c))^2}{3mnl} \right)}$$

where $P_1(x, y, z, c)$ means that the pixel value at coordinates (x, y) in the z-th frame for color channel c, where $c = 1$, $c = 2$, and $c = 3$ corresponds to the color channels red, green, and blue, respectively. In our experiment, the values of m,n, and l are 640, 480 and 1000 respectively. Values of P_1 are obtained from the video frames decoded by a client which does not have access to the decryption key, while values of P_2 are obtained from the original video frames. Note that a lower value of SNR_{peak} indicates that the encrypted stream is more distorted from the original video stream.

Tables 3 and 4 illustrate the peak signal-to-noise ratio SNR_{peak} for different values of E_p and E_i with $E_b = 1$ for MPEG-1 and Quicktime video, respectively. The experiment result shows that the value of SNR_{peak} is proportional to the amount of data we encrypt, which can be controlled by adjust E_p and/or E_i, for both MPEG1 and Quicktime video. For example, if we only want to encrypt 4.3% of the whole video stream, we can choose either $E_p = 0.043$ and $E_i = 1$, or $E_p = 0.086$ and $E_i = 2$, or $E_p = 0.171$ and $E_i = 5$, where each ECP combination will produce a similar SNR_{peak} value. If we double the amount of data to be encrypted (i.e. 8.6% of the whole video stream), we can choose either $E_p = 0.086$ and $E_i = 1$, or $E_p = 0.171$ and $E_i = 2$ and either ECP combination will product a similar SNR_{peak} value that implies a more distorted video than encrypts 4.3% of the whole

Table 3: Effect of E_p and E_i on the peak signal-to-noise ratio SNR_{peak} on MPEG-1 video when $E_b = 1$

	peak signal-to-noise ratio $\mathbf{SNR_{peak}}$					
$\mathbf{E_i \backslash E_p}$	0.257	0.214	0.171	0.120	0.086	0.043
1	9.5966	9.9205	10.124	10.215	10.419	10.238
2	8.8358	10.106	10.419	10.164	10.133	10.545
5	8.2106	8.1195	9.4196	9.7872	8.3017	9.9224
10	10.600	12.317	11.426	9.9980	8.9169	12.278

Table 4: Effect of E_p and E_i on the peak signal-to-noise ratio SNR_{peak} on Quicktime video when $E_b = 1$

	peak signal-to-noise ratio $\mathbf{SNR_{peak}}$					
$\mathbf{E_i \backslash E_p}$	0.257	0.214	0.171	0.120	0.086	0.043
1	11.774	12.053	12.340	12.552	12.970	13.397
2	12.509	12.704	12.939	13.084	13.397	13.875
5	13.298	13.396	13.636	13.953	14.175	14.727
10	13.953	13.878	14.177	14.517	14.728	14.973

video stream. Therefore, one can expect the degree of data confidentiality (in terms of SNR_{peak}) decreases proportionally when decreasing the amount of data to protect in order to trade-off for a higher throughput, and without the need to take care of the video's encoding format. Note that even when we encrypt one out of ever 10 video packets, and for the selected packets, we only encrypt 4.3% of the data, we can still obtain a very low value of SNR_{peak}. This experiment indicates that (1) we can apply this encryption technique for different video formats (e.g., MPEG1 or Quicktime) and, (2) we only need to encrypt a small fraction of the video data to achieve *both* high encryption throughput and high video distortion.

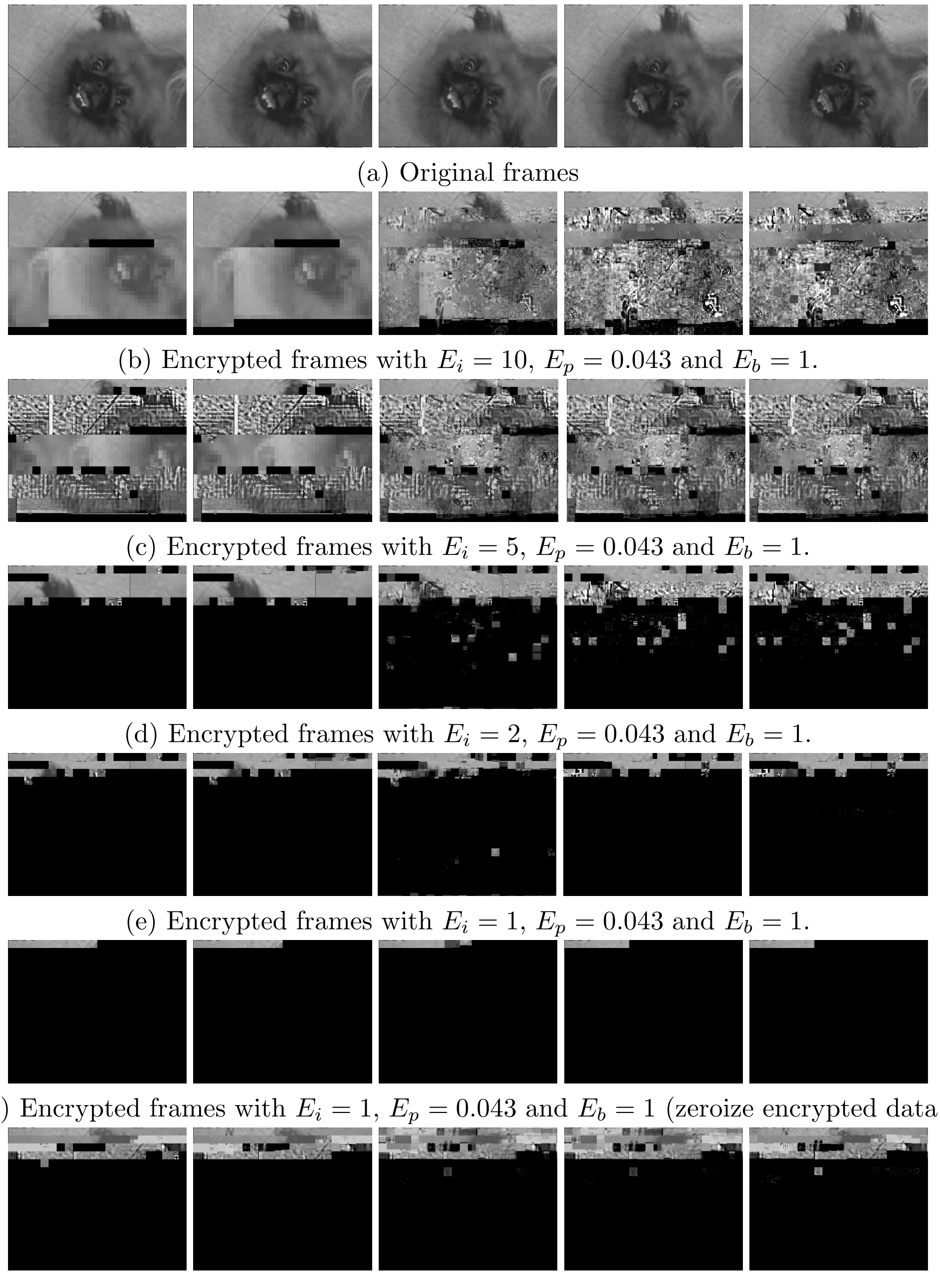
Table 5 illustrates the effect of varying E_i and E_b under two different encryption percentage parameter E_p. As we can observe, the parameter E_b has little effect on the peak signal-to-noise ratio $\mathbf{SNR_{peak}}$.

Experiment 3 (Comparison of visual quality of encrypted video):
In this experiment, we consider the effect of varying the ECP parameters E_i, E_p and E_b on the *visual quality* of the video. Figure 4 illustrates the quality of five consecutive MPEG-1 video frames. Figure 4(a) is the original video frames that a client can decode given access to the decryption key. Figures 4(b)-(e) are the corresponding five video frames when decoded without the decryption key. Note that the video quality is the worst when the ECP parameters are set to $E_i = 1$ and $E_p = 0.043$, which corresponds to encrypting 4.3% of the data for every video packet (this corresponds to Figure 4(e)), the original content of the video is nearly indiscernible. The visual quality of the decoded video improves gradually from $E_i = 1$ to $E_i = 10$ while the value of E_p and E_b is not changed. Note that when we select $E_i = 10$, $E_p = 0.043$ (this corresponds to Figure 4(b)), the visual quality of the video is still unacceptable for viewing. This shows that we can achieve high encryption throughput (i.e., around 11.82 MBytes/s or about 63 concurrent MPEG-1 streams from Table 1) and, at the same time, ensure that those clients which do not possess the decryption keys will get an unacceptable video quality on viewing. Therefore, one may choose a lightweight ECP combination (e.g. $E_i = 10$, $E_p = 0.043$) for a casual video-on-demand service in order to support more concurrent clients, but one may choose a heavyweight ECP combination (e.g. $E_i = 1$, $E_p = 0.043$) for a private video conferencing session in order to achieve the maximum affordable data confidentiality. Figure 5 shows the corresponding results for five consecutive Quicktime video frames. Similar conclusions can be drawn from the Quicktime results.

Experiment 4 (Discarding encrypted data analysis):
In this experiment, we consider the effect on the video quality when an unauthorized party just try to discard all of the encrypted data before decoding an encrypted stream without having the proper decryption key. We consider two different ways to discard those encrypted data. The first one is to drop all of the encrypted data, and the second one is to fill all of the encrypted data with zeros.

Table 6 illustrates the peak signal-to-noise ratio SNR_{peak} for dropping encrypted data and filling encrypted data with zeros under four different ECP encryption schemes. Note that we get similar, or even lower values of $SNRpeak$ when discarding encrypted data, compared to direct decoding of the encrypted streams. Figure 4(f-g) and 5(f-g) show the five video frames decoded in each of the streams respectively, they suggested that discarding encrypted data *does not* help in improving the visual quality. This experiment indicates that an unauthorized party cannot get a better decoding quality by means of discarding the encrypted video data.

Experiment 5 (Signal-to-Noise Analysis for Audio Streaming Application):
In this experiment, we consider the effect on the audio quality as we vary the parameters E_i, E_p and E_b. The audio clip used in this experiment is a MPEG-1 layer 3 (or MP3) [6] audio file encoded at a bit-rate of 128 kb/s. We compute the signal-to-noise ratio (SNR) with a Matlab



(a) Original frames

(b) Encrypted frames with $E_i = 10$, $E_p = 0.043$ and $E_b = 1$.

(c) Encrypted frames with $E_i = 5$, $E_p = 0.043$ and $E_b = 1$.

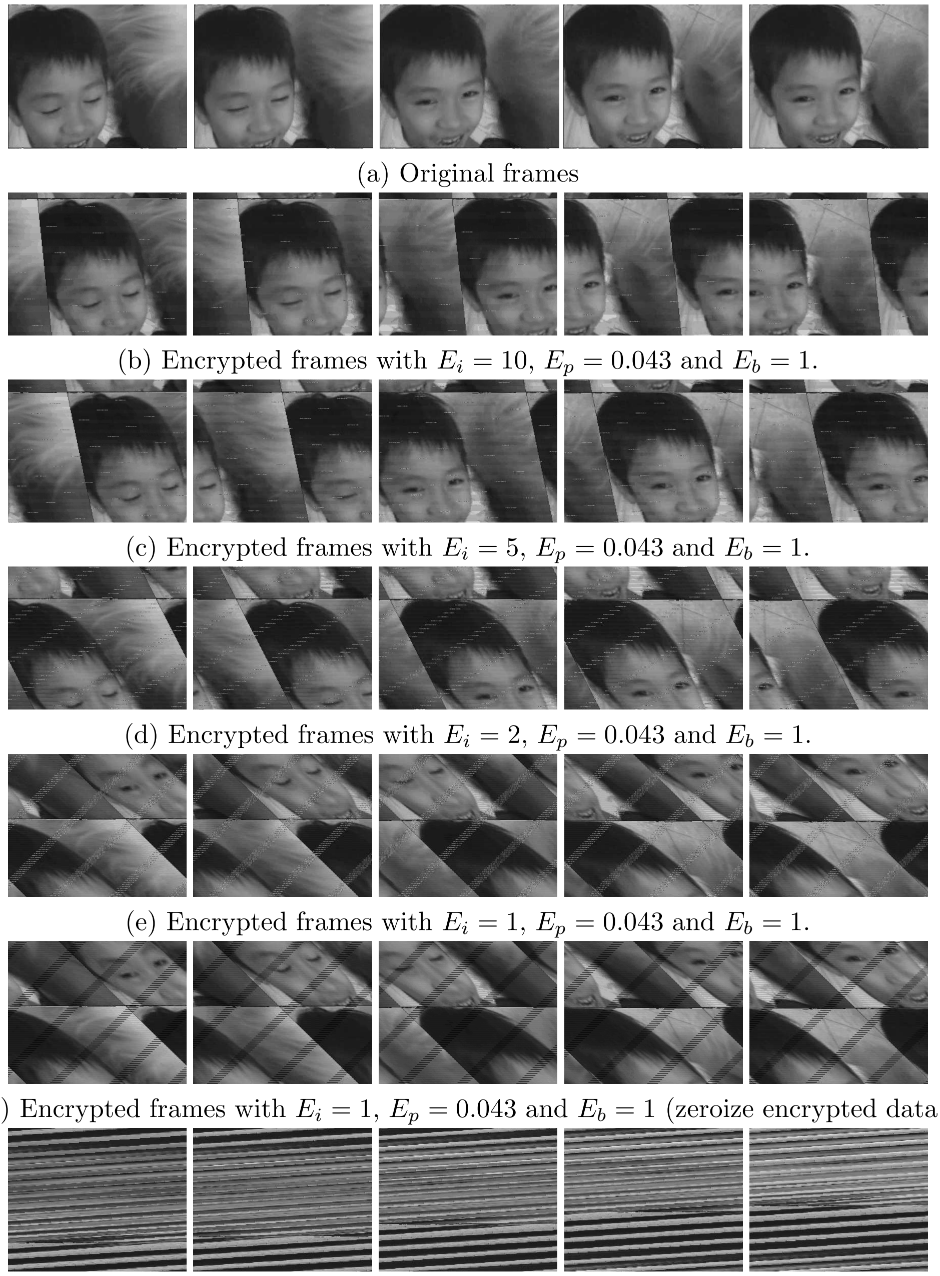
(d) Encrypted frames with $E_i = 2$, $E_p = 0.043$ and $E_b = 1$.

(e) Encrypted frames with $E_i = 1$, $E_p = 0.043$ and $E_b = 1$.

(f) Encrypted frames with $E_i = 1$, $E_p = 0.043$ and $E_b = 1$ (zeroize encrypted data.)

(g) Encrypted frames with $E_i = 1$, $E_p = 0.043$ and $E_b = 1$ (drop encrypted data.)

Figure 4: Quality of five consecutive MPEG-1 video frames under different ECP parameters



(a) Original frames

(b) Encrypted frames with $E_i = 10$, $E_p = 0.043$ and $E_b = 1$.

(c) Encrypted frames with $E_i = 5$, $E_p = 0.043$ and $E_b = 1$.

(d) Encrypted frames with $E_i = 2$, $E_p = 0.043$ and $E_b = 1$.

(e) Encrypted frames with $E_i = 1$, $E_p = 0.043$ and $E_b = 1$.

(f) Encrypted frames with $E_i = 1$, $E_p = 0.043$ and $E_b = 1$ (zeroize encrypted data.)

(g) Encrypted frames with $E_i = 1$, $E_p = 0.043$ and $E_b = 1$ (drop encrypted data.)

Figure 5: Quality of five consecutive Quicktime video frames under different ECP parameters

Table 5: Effect of E_i and E_b on the peak signal-to-noise ratio $\mathbf{SNR_{peak}}$ for (a) $E_p = 0.257$ and (b) $E_p = 0.171$

	peak signal-to-noise ratio $\mathbf{SNR_{peak}}$			
$\mathbf{E_i \backslash E_p}$	1	2	5	10
1	11.77	12.51	13.30	13.95
2	11.89	12.52	13.30	13.95
3	12.01	12.50	13.29	13.95

(a) $E_p = 0.257$

	peak signal-to-noise ratio $\mathbf{SNR_{peak}}$			
$\mathbf{E_i \backslash E_p}$	1	2	5	10
1	12.34	12.94	13.64	14.18
2	12.53	12.95	13.63	14.18
4	12.79	12.96	13.63	14.18

(b) $E_p = 0.171$

Table 6: Effect of discarding encrypted data on the peak signal-to-noise ratio SNR_{peak} when $E_b = 1$ and $E_p = 0.043$

	$\mathbf{SNR_{peak}}$		
	direct	**drop**	**fillzero**
$\mathbf{E_i = 1}$	10.24	9.10	8.33
$\mathbf{E_i = 2}$	10.55	9.10	8.97
$\mathbf{E_i = 5}$	9.92	11.35	10.56
$\mathbf{E_i = 10}$	12.28	12.76	12.76

(a) MPEG-1 streams

	$\mathbf{SNR_{peak}}$		
	direct	**drop**	**fillzero**
$\mathbf{E_i = 1}$	13.40	13.36	13.27
$\mathbf{E_i = 2}$	13.88	13.67	13.81
$\mathbf{E_i = 5}$	14.73	13.40	14.70
$\mathbf{E_i = 10}$	14.97	13.58	14.95

(b) QuickTime streams

(**direct**: decode directly;
drop: drop encrypted data;
fillzero: fill encrypted data with zeros.)

program using the following equation:

$$SNR = \frac{\sum_{i=1}^{n} original(i)^2}{\sum_{i=1}^{n}(original(i) - cipher(i))^2}$$

where $original(i)$ denotes the i-th sample in the waveform decoded from the original audio stream, and $cipher(i)$ denotes the i-th sample in the waveform decoded from the encrypted audio stream without the decryption key. In this experiment, n equals to 44100, which means that samples from the first second of the audio stream are used. Note that a lower value of SNR indicates that the encrypted audio stream is acoustically more distorted from the original audio stream, while an SNR value of *infinity* indicates that the measured samples are exactly identical to those in the original audio stream.

Table 7 illustrates the signal-to-noise ratio SNR for different values of E_i and E_p, when $E_b = 1$. Again, we observe that one does not need to encrypt all the audio packets to sufficiently distort the audio signal. In general, our proposed ECP method allows one to simultaneously achieve high encryption throughput and low audio fidelity during unauthorized access.

6 Conclusion

We present the design and implementation of a secure RTSP multimedia streaming architecture that enables secure and hierarchical proxy caching. Our design is based on the notion of an *asymmetric reversible parametric sequence* (ARPS). We discussed how ARPS could be applied to general client-proxy-server architecture and we provided the tools required to realize ARPS as a C language API, the SML. To demonstrate the usefulness of our system model, we have described in detail an extended RTSP with ARPS integration that can provide secure proxy caching and meanwhile compatible to standard RTSP system. We have implemented such a secure RTSP multimedia streaming system consisting of an RTSP server, an RTSP proxy and an RTSP client. Our experimental results empirically demonstrated how a set of four ECP parameters can be used to trade off encryption throughput against the amount of data to protect, for a number of standard MPEG-1 and QuickTime video sequences, and a number of MP3 audio sequences. Our results indicate that it is possible to *simultaneously* achieve high encryption throughput and extremely low audio/video quality (in terms of decoded audio SNR and both PSNR and the visual quality of decoded video frames) during unauthorized accesses.

References

[1] Apple Computer Inc., *Darwin Streaming Server 4.1.3*, 2003. (http://developer.apple.com/darwin/projects/streaming/)

[2] I. E. T. Force, *Public-Key Infrastructure (X.509)*, 2003. (http://www.ietf.org/html.charters/

Table 7: Effect of E_p and E_i on the signal-to-noise ratio SNR on MP3 audio when $E_b = 1$

	signal-to-noise ratio SNR				
	$\mathbf{E_p = 0.214}$	$\mathbf{E_p = 0.171}$	$\mathbf{E_p = 0.120}$	$\mathbf{E_p = 0.086}$	$\mathbf{E_p = 0.043}$
$\mathbf{E_i = 1}$	0.9104	0.7720	0.8571	0.8429	0.8264
$\mathbf{E_i = 2}$	0.5831	0.5608	0.5614	0.5585	0.5707
$\mathbf{E_i = 5}$	0.5479	1.0334	1.0360	13.6172	2.3095
$\mathbf{E_i = 10}$	1.0494	1.0494	1.0494	25.1848	25.1849

pkix-charter.html)

[3] C. Griwodz, O. Merkel, J. Dittmann, and R. Steinmetz, "Protecting vod the easier way," *Proceeding of the 6th ACM International Multimedia Conference*, pp. 21–28, 1998.

[4] S. Gruber, J. Rexford, and A. Basso, "Protocol considerations for a prefix-caching proxy for multimedia streams," *Computer Networks*, vol. 33, no. 1–6, pp. 657–668, 2000.

[5] Y. Guo, S. Sen, and D. Towsley. "Prefix caching assisted periodic broadcast: Framework and techniques to support streaming for popular videos," *IEEE International Conference on Communications (IEEE ICC 2002)*, vol. 4, pp. 2607–2612, Marriott Marquis, New York, 2002.

[6] International Organisation for Standardisation, *Short MPEG-1 description*, 1996. (http://mpeg.telecomitalialab.com/standards/mpeg-1/mpeg-1.htm)

[7] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross, "Distributing layered encoded video through caches," *Proceedings of IEEE INFOCOM 2001*, pp. 1791–1800, Anchorage, Alaska, 2001.

[8] R. Molva and A. Pannetrat, "Scalable multicast security in dynamic groups," *Proceeding of the 6th ACM Conference on Computer and Communications Security (CCS1999)*, pp. 101–111, 1999.

[9] RealNetworks. *RTSP Proxy Kit 2.0*, 2001. (http://www.rtsp.org/2001/proxy)

[10] R. Rejaie, M. Handley, H. Yu, and D. Estrin, "Proxy caching mechanism for multimedia playback streams in the internet," *Proceedings of the 4th International Web Caching Workshop*, San Diego, CA., 1999.

[11] S. Sen and D. Towsley, "Proxy prefix caching for multimedia streams," *Proceedings of IEEE INFOCOM 1999*, pp. 1310–1319, New York, 1999.

[12] D. K. Y. S.F. Yeung, John C.S. Lui, "A multi-key secure multimedia proxy using asymmetric reversible parametric sequences: Theory, design, and implementation," *IEEE Transactions on Multimedia*, vol. 7, no. 2, pp. 330–338, 2005.

[13] C. Shi and B. Bhargava, "A fast mpeg video encryption algorithm," *Proceeding of the 6th ACM International Multimedia Conference*, pp. 81–88, Bristol, United Kingdom, 1998.

[14] C. Systems, *MPEG4IP*, 2003. (http://mpeg4ip.sourceforge.net/index.php)

[15] A. S. Tosun and W. chi Feng, "Secure video transmission using proxies," Technical Report, Computer and Information Science, Ohio State Univeristy, 2002.

Yeung Siu Fung is a Ph.D. candidate at the Chinese University of Hong Kong. He received his B.Eng. and M.Phil. degree in the Computer Science and Engineering Department from the Chinese University of Hong Kong. His research are in multimedia technologies, particularly network security and transport protocols. His personal interests include sports and Christian music.

John C. S. Lui received his Ph.D. in Computer Science from UCLA. After his graduation, he joined the IBM Almaden Research Laboratory/San JoseLaboratory and participated in various research and development projects on file systems and parallel I/O architectures. He later joined the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research interests encompass both systems and theory. His current research interests include theoretic/applied topics in data networks, dis-tributed multimedia systems, network security, OS design issues, mathematical optimization and performance evaluation. John received the CUHK Vice-Chancellor's Exemplary Teaching Award in 2001. He is an Associate Editor ofthe Performance Evaluation Journal, a member of the ACM, a senior member of the IEEE and an elected member of the IFIP WG 7.3. His personal interests include films and general reading.