A Multi-key Secure Multimedia Proxy Using Asymmetric Reversible Parametric Sequences: Theory, Design, and Implementation

Siu F. Yeung, John C. S. Lui, *Senior Member, IEEE*, David K. Y. Yau, *Member, IEEE*

***Abstract*— Because of limited server and network capacities for streaming applications, multimedia proxies are commonly used to cache multimedia objects such that, by accessing nearby proxies, clients can enjoy a smaller start-up latency and receive a better quality-of-service (QoS) guarantee – for example, reduced packet loss and delay jitters for their requests. However, the use of multimedia proxies increases the risk that multimedia data are exposed to unauthorized access by intruders. In this paper, we present a framework for implementing a secure multimedia proxy system for audio and video streaming applications. The framework 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. Our framework is grounded on a *multi-key RSA* technique such that system resilience against attacks is provably strong given standard computability assumptions. One important feature of our proposed scheme is that clients only need to perform a single decryption operation to recover the original data even though the data packets may have been encrypted by multiple proxies along the delivery path. 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. Implementation results show that we can *simultaneously* achieve high encryption throughput and extremely low video quality (in terms of peak signal-to-noise ratio and visual quality of decoded video frames) for unauthorized access.**

***Index Terms*— Security, Asymmetric Parametric Sequence Functions, Multi-Key RSA, Video Proxy**

I. Introduction

Advancements in digital audio/video and compression technologies have led to the recent wide deployment of continuous media streaming over the Internet. To make the video streaming service scalable, one can use a multimedia proxy to perform some form of data caching of popular audio/video objects, so that clients can access the cached data from their nearby proxies to reduce startup delay and conserve bandwidth. One major problem with the multimedia proxy approach is the risk of revealing the original multimedia data to unauthorized parties. For example, when the original multimedia data are sent from a server to a proxy, anyone that eavesdrops on the communication link between the source and the proxy can gain access to the audio/video information.

In this paper, we present a proxy encryption framework having the following security properties:

- The multimedia proxy will only cache *encrypted* multimedia data and data decryption will only happen at the endpoints (i.e, clients). Therefore, the original multimedia data will not be revealed at any intermediate node.
- The multimedia proxy will perform encryption operations only. This reduces the computational overhead at the proxy, and hence allows one to build a more scalable proxy system.
- Data encryption and decryption operations are based on well accepted encryption theory that it is computationally infeasible to extract the original multimedia data.
- *Member collusion* can be avoided, such that given (1) the decryption key of client i, (2) the encrypted data of client j, and (3) possibly all the encryption keys, the intruder still cannot derive the original multimedia data.
- The proposed approach can be extended to a multi-level proxy structure. This is an important property since it is common to have multiple proxies along the communication path between the sender and a receiver.

The rest of the paper is organized as follows. In Section II, we present multi-key encryption based on the *asymmetric reversible parametric sequence*. We then present a practical algorithm to implement an asymmetric reversible parametric sequence to achieve the claimed security properties for a multimedia proxy server. In Section III, we present our proxy system architecture and the proxy-server and client-proxy communication protocols. In Section IV, we report implementation results that illustrate the achievable encryption data rate using our technique on a commodity Pentium machine, and give quantitative and qualitative analysis of the encrypted audio/video quality. Related work on multimedia proxies is presented in Section V. Section VI concludes.

II. Multi-Key Encryption Theory

In this section, we state the theory behind the design of a multi-key secure video proxy. The main theory is based on the reversible parametric sequence (RPS) [7]. In [7], Molva *et al.* use RPS to perform group key management for multicast security. In this paper, we will explore the efficient use of RPS on bulk multimedia data. We will discuss how to use the idea of RPS to construct a framework for secure video/audio proxy streaming. In the following, we first present the formal definition of RPS and its utilities. We then present an implementation of RPS using the multi-key RSA technique.

A. Reversible Parametric Sequence (RPS)

Let $f : IN^2 \rightarrow IN$ be a function which has the following property: if $Y = f(X, e)$, it is *computationally infeasible* to find e given that we know X and Y.

Assume that we have a finite sequence $\{e_0, e_1, \cdots, e_N\}$ of $N+1$ elements, where the elements are not necessarily unique. Define a finite data transformation sequence $\mathcal{D} = \{D_{-1}, D_0, ..., D_N\}$ based on the function f and the finite sequence $\{e_i\}_{0 \le i \le N}$ such that

$$\begin{aligned} D_{-1} &= \text{original data} \\ D_i &= f(D_{i-1}, e_i) \quad \text{for } 0 \le i \le N. \end{aligned}$$

We have the following definitions.

Definition *1:* $\mathcal{D}$ is a *reversible parametric sequence* of the function f, denoted as RPS_f, if for all $(X, Y) \in I\!N^2$ and $-1 \le i < j \le N$, there exists a computable function $\Omega_{i,j}$ such that $D_i = \Omega_{i,j}(D_j)$, for $-1 \le i < j \le N$.

Definition *2:* An RPS_f is called a *symmetric reversible parametric sequence* of f, denoted as $SRPS_f$, if the function $\Omega_{i,j}$ can be computed from the knowledge of the sub-sequence $\{e_{i+1}, \cdots, e_j\}$.

Definition *3:* An RPS_f is called an *asymmetric reversible parametric sequence* of f, denoted as $ARPS_f$, if it is computationally infeasible to determine the function $\Omega_{i,j}$ based on the knowledge of the sub-sequence $\{e_{i+1}, \cdots, e_j\}$.

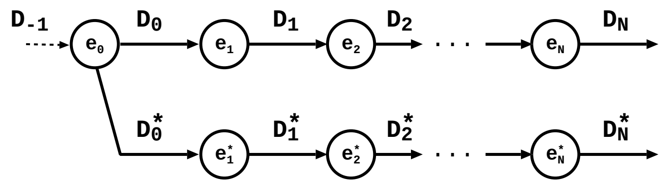


Fig. 1. A graphical representation of two RPS_f sequences with $D_i \ne D_i^*$.

To illustrate, we use a graph to represent a reversible parametric sequence RPS_f. Figure 1 illustrates two RPS_f sequences. In the figure, the original data D_{-1} are transformed to D_0 using $D_0 = f(D_{-1}, e_0)$. If $e_i \ne e_i^*$, then the intermediate data D_i will not be equal to D_i^*, for $1 \le i \le N$. For a symmetric reversible parametric sequence $SRPS_f$, one *can* compute the original data D_{-1} if given the data D_j (or D_j^*) and the sequence $\{e_0, \cdots, e_j\}$ (or $\{e_0^*, \cdots, e_j^*\}$), for $0 \le j \le N$. In other words, given the information $\{e_0, \ldots, e_j\}$ and D_j, one can construct a decryption function $\Omega_{-1,j}$ so as to obtain the original data D_{-1}. For an asymmetric reversible parametric sequence $ARPS_f$, one *cannot* derive the original data D_{-1} even if given the data D_j (or D_j^*) and the knowledge of the sequence $\{e_0, \cdots, e_j\}$ (or $\{e_1^*, \cdots, e_j^*\}$), for $0 \le j \le N$.

One can use the properties of an asymmetric reversible parametric sequence $ARPS_f$ to implement a secure multimedia proxy. To illustrate, consider a graphical representation of an $ARPS_f$ sequence in Figure 2. The multimedia proxy can request D_0, the encrypted version of the original data D_{-1}, from the source. Based on an encrypted key e_0, the source will transmit the encrypted data D_0 to the proxy, and the encrypted data D_0 will be cached at the proxy's local storage. When a client i requests the data, the proxy will further encrypt the encrypted data D_0 using the encryption key e_i and send the resulting encrypted data D_i to client i. Client i, upon receiving D_i, can decrypt the data to obtain the original data

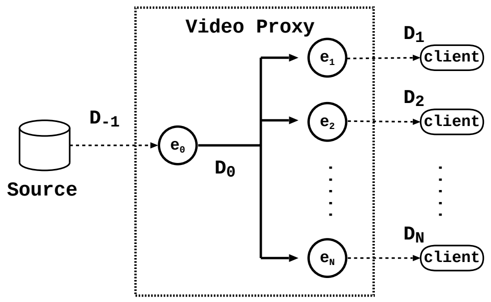


Fig. 2. A graphical representation of an $ARPS_f$ sequence for the secure multimedia proxy.

D_{-1}, if client i is given a decryption function $\Omega_{-1,i}$ (this is a property of reversible parametric sequences). In addition, when the encryption is carried out using an asymmetric reversible parametric sequence, then even when an entity holds on to *all* the encryption keys e_i for $0 \le i \le N$, it still cannot decrypt any of the encrypted data D_i being cached, for $0 \le i \le N$, in order to obtain the original data D_{-1}.

In general, one can use an asymmetric reversible parametric sequence $ARPS_f$ to implement a secure multimedia proxy which has the following properties: (1) data confidentiality during transmission, (2) end-to-end data confidentiality, (3) data confidentiality against proxy intruders, and (4) data confidentiality against member collusion. Let us elaborate on the last point. If the encryption process is $SRPS_f$, member collusion is possible when a client j gains access to

1) e_i and e_j (where $i, j > 0$),
2) the encrypted data D_i, and
3) the decrypting function $\Omega_{-1,j}$.

In this case, client j (i.e., the intruder) can access the original data D_{-1}. For example,

1) Given e_i, the intruder can obtain $\Omega_{0,i}$ and thus obtain $D_0 = \Omega_{0,i}(D_i)$.
2) Given the knowledge of e_j and D_0, the intruder can obtain D_j by $D_j = f(D_0, e_j)$.
3) Since the intruder knows the decryption function $\Omega_{0,j}$, the original data D_{-1} are revealed by $D_{-1} = \Omega_{-1,j}(D_j)$.

However, if the encryption process is an $ARPS_f$, the intruder cannot reveal the original data D_{-1} because the function $\Omega_{0,i}$ is computationally infeasible to determine.

B. Implementation of $ARPS_f$

To realize an ARPS function, one needs a practical and efficient algorithm. To this end, we present an extension of single-key RSA [10] to a multi-key RSA technique.

Consider the source as an example. The source needs to generate two large prime numbers, say p and q. In addition, it needs to generate a sequence of encryption keys $\{e_i\}_{0 \le i \le N}$ such that

$$1 < e_i < \phi \qquad \text{and} \qquad (1)$$

$$gcd(e_i, \phi) = 1 \qquad \text{for } 0 \le i \le N. \qquad (2)$$

Moreover, one needs to generate a corresponding sequence of decryption keys $\{d_i\}$. Each decryption key d_i has to satisfy the following two criteria:

$$1 < d_i < \phi \quad \text{and} \tag{3}$$
$$(e_0 \cdot e_i) \cdot d_i = 1 \text{ (mod } \phi). \tag{4}$$

Efficient computation of these decryption keys d_i can be easily achieved using the Extended Euclidean Algorithm [9]. The source will send n and the encryption keys e_i over a secure channel to the multimedia proxy, while it will encrypt the original data D_{-1} using e_0 and generate a cipher D_0 using

$$D_0 \quad = \quad (D_{-1})^{e_0} \text{mod } n. \tag{5}$$

The encrypted data D_0 can be sent over an insecure channel. Upon receiving the cipher D_0, the proxy can store the encrypted data in its local storage. Whenever a client i wishes to access the data from the proxy, the proxy can retrieve the encrypted data D_0 from its local storage, and encrypt D_0 using the encryption key e_i by

$$D_i \quad = \quad (D_0)^{e_i} \text{mod } n. \tag{6}$$

The source (or the proxy) can send the decryption key d_i and n to client i over a secure channel. The encrypted data D_i, on the other hand, can be sent over an insecure channel. Client i, upon receiving the encrypted data D_i, can decrypt the data using d_i by:

$$D_{-1} \quad = \quad \Omega_{-1,i}(D_i) = (D_i)^{d_i} \text{mod } n. \tag{7}$$

Theorem *1:* The above proxy encryption framework is a reversible parametric sequence.

Proof: To show that the above framework is a reversible parametric sequence, we need to show that given D_i, for $i \geq 1$, we can extract D_0 and D_{-1}. Without loss of generality, let us consider D_1 as the given input. The generation of D_1 is via

$$\begin{aligned} D_1 &= (D_0)^{e_1} \text{mod } n. = \left[(D_{-1})^{e_0} \text{mod } n\right]^{e_1} \text{mod } n. \\ &= (D_{-1})^{e_0 \cdot e_1} \text{mod } n. \end{aligned}$$

Let the extracted result be R and equal to

$$\begin{aligned} R &= (D_1)^{d_1} \text{mod } n = \left[(D_{-1})^{e_0 \cdot e_1} \text{mod } n\right]^{d_1} \text{mod } n \\ &= (D_{-1})^{e_0 \cdot e_1 \cdot d_1} \text{mod } n. \end{aligned}$$

Since the encryption key e_0 and the decryption key d_1 are generated such that $e_0 \cdot e_1 \cdot d_1 = k \cdot (p-1) \cdot (q-1) + 1$, where k is an integer, we can rewrite the extracted result R as

$$R = \left[(D_{-1})(D_{-1})^{k(p-1)(q-1)}\right] \text{mod } n.$$

Based on Euler's theorem [9] that $X^{p-1} = 1 \,\text{mod}\, n$, we have the following expressions:

$$R = (D_{-1})(D_{-1})^{k(p-1)(q-1)} = (D_{-1})1^{k(q-1)} = D_{-1}\text{mod } p,$$
$$R = (D_{-1})(D_{-1})^{k(p-1)(q-1)} = (D_{-1})1^{k(p-1)} = D_{-1}\text{mod } q.$$

Because p and q are primes, based on the Chinese Remainder Theorem [9], we have

$$R = (D_{-1})(D_{-1})^{k(p-1)(q-1)} = D_{-1}(\text{mod } n) = D_{-1}.$$

Therefore, given D_1, one can extract D_{-1} given the knowledge of d_1. We can apply a similar procedure such that given D_0, one can extract D_1 with another corresponding decryption key. In summary, given D_i, one can extract D_0 and the original data D_{-1} if the corresponding decryption key is known. ∎

Theorem *2:* The above proxy encryption procedure is an asymmetric reversible parametric sequence.

Proof: To show that the above framework is an asymmetric reversible parametric sequence, all we need to show is that given the encrypted data D_i, for $i \geq 0$, it is impossible to obtain the original data D_{-1}, even if one has the knowledge of $\{e_0, e_1, e_N\}$ and n. Since we use RSA encryption, finding the decryption keys in Equations (3)–(4) amounts to finding the two prime factors p and q, which is well accepted to be computationally infeasible if p and q are large and properly chosen. Therefore, it is computationally infeasible to find $\Omega_{i,j}$. ∎

III. Proxy Architecture and Protocols

In this section, we describe in detail our server-proxy-client architecture. The multimedia server $\mathcal{S}$, the multimedia proxy $\mathcal{P}$, and various clients have been implemented in the C language on a Linux platform. Security features such as key generation, encryption, and decryption are implemented using the GNU Multiply Precision (GMP 4.0) library, which provides arbitrary precision arithmetic on integers, rational numbers, and floating-point numbers. Note that GMP 4.0 provides one of the fastest possible arithmetic libraries for applications that need higher precision than what is directly supported by the basic C types.

A. Operations to Request and Cache Data from the Server

Let us first consider the case wherein the multimedia data are not yet cached at the proxy. Figure 3a illustrates the operations between the multimedia server $\mathcal{S}$ and the proxy $\mathcal{P}$, and the operations between the proxy $\mathcal{P}$ and the client i for requesting and caching the multimedia data. These operations are:

(1) **Initiate connection:** The client i sends a multimedia request and a *certificate* of its identity to the proxy $\mathcal{P}$. The proxy $\mathcal{P}$ forwards the request to the multimedia server $\mathcal{S}$. [Client i's certificate is $eCert_i = (u_i, \{u_i\}_{v_i})$, where u_i and v_i are i's public and private keys, respectively, for RSA cryptography, and $\{D\}_k$ denotes information D encrypted with key k.] The multimedia server keeps a record of the public key for each of its authenticated clients. To verify the identity of the requesting client, the multimedia server $\mathcal{S}$ decrypts $\{u_i\}_{v_i}$ using v_i as the decryption key. The request will only be granted if the decrypted message equals u_i and matches the public key of i in the server's authorization list. This relies on the fact that only the genuine client i has the secret key v_i. Hence, only client i can generate the $\{u_i\}_{v_i}$ which, when decrypted using v_i, will produce the same u_i as the copy stored by the server. A random server challenge will

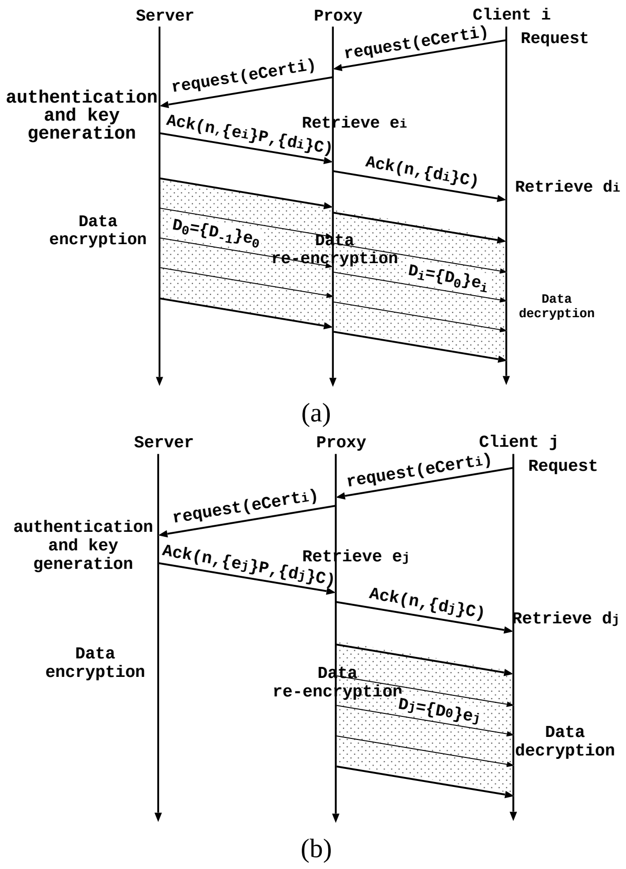


Fig. 3. Operations between the source video server $\mathcal{S}$ and the proxy $\mathcal{P}$, and operations between the proxy $\mathcal{P}$ and the client i, for (a) non-cached video object and (b) cached video object. Note that P is the public key of the proxy and C is the public key of the client, and $\{D\}_k$ denotes information D encrypted with key k.

then be performed. The challenge eliminates the possibility of a fake client using another client's certificate obtained by eavesdropping. The server generates a random message, sends it to proxy $\mathcal{P}$, which in turn sends it to client i. Client i encrypts the message using its own private key and replies to proxy $\mathcal{P}$. Proxy $\mathcal{P}$ also encrypts the message using its own private key, and includes the encrypted message in its reply. Finally, server $\mathcal{S}$ decrypts the challenge-replies using the public keys of client i and proxy $\mathcal{P}$, respectively. A request will only be granted if the decrypted messages match the challenge message. If the authentication is successful, then the server $\mathcal{S}$ will proceed to the key generation operation.
(2) **Key generation:** The server randomly generates two large prime numbers p and q, and computes the modulus $n = p \cdot q$, $\phi = (p-1)\cdot(q-1)$, the encryption key e_0, re-encryption key e_i, and the corresponding decryption key d_i via Equations (1) and (2)[1]. The server $\mathcal{S}$ saves the parameters ϕ, e_0 and n with a unique identifier ID. It then replies back to the proxy $\mathcal{P}$ with (1) the re-encryption key e_i, which is encrypted using the proxy's public key, and (2) the corresponding decryption key d_i, which is encrypted using the client i's public key. Note that the proxy $\mathcal{P}$ cannot extract the decryption key d_i, but only the client i can perform the decryption to extract d_i. And since the re-encryption key e_i is encrypted using the proxy's public key, no one can reveal it by eavesdropping on the channel between the server and the proxy. Collusion attacks can thus be avoided. Consider two colluding members knowing the re-encryption keys e_i and e_j (by eavesdropping) and decryption keys d_i and d_j, it can be shown that they can compute the secret parameter ϕ efficiently [13]. Once the secret parameter ϕ is revealed, the colluding members can derive the proxy's decryption key or any other client's decryption key. Hence, the re-encryption key e_i must be encrypted such that only the proxy can retrieve it.
(3) **Decryption key retrieval:** The proxy $\mathcal{P}$ replies with an acknowledgment back to the client i including the encrypted d_i. The client i decrypts using its own private key to retrieve the decryption key d_i.
(4) **Data encryption and streaming:** The multimedia server $\mathcal{S}$ uses the encryption key e_0 and n to encrypt the multimedia data packets. The degree of encryption is based on the encryption configuration parameters (ECP) which will be described in Section III-C. The multimedia server $\mathcal{S}$ then streams the encrypted data packets to the proxy $\mathcal{P}$ via an ordinary and possibly insecure channel. Upon receiving the encrypted data packets, the proxy $\mathcal{P}$ caches the data without decryption or modification. (Note that the data packets may require proper formatting, such as padding, before the encryption. For simplicity, these issues are omitted in this paper. Readers may refer to [5] or [1] for more information.
(5) **Data re-encryption and streaming:** The proxy $\mathcal{P}$ uses the re-encryption key e_i and n to re-encrypt the already encrypted multimedia data packets in the cache. The encryption is based on the same ECP setting used by server $\mathcal{S}$. The proxy $\mathcal{P}$ then streams the re-encrypted data packets to the client i via an ordinary and possibly insecure channel. Upon receiving the re-encrypted data packets, the client i can use the decryption key d_i to decrypt the received multimedia data packets.

B. Operations to Request Cached Data from the Proxy

We consider the case wherein the multimedia data are already cached at the proxy. Figure 3b illustrates the operations between the multimedia server $\mathcal{S}$ and the proxy $\mathcal{P}$, and the operations between the proxy $\mathcal{P}$ and the client j for requesting multimedia data that are already cached by the proxy $\mathcal{P}$. These operations are:
(1) **Initiate connection:** The client j sends a request to the proxy $\mathcal{P}$ with its certificate. The proxy $\mathcal{P}$ forwards the request to the server $\mathcal{S}$ with a specified unique identifier ID. The multimedia server $\mathcal{S}$ needs to authenticate that the request is indeed from the client j. If the authentication succeeds, the server $\mathcal{S}$ will go on to the key generation operation.
(2) **Key generation:** The server $\mathcal{S}$ randomly generates a re-encryption key e_j and a corresponding decryption key d_j based on the ϕ, n and e_0 identified by ID. It then sends back to the proxy $\mathcal{P}$ (i) the re-encryption key e_j, which is encrypted using the proxy's public key, and (ii) the decryption key d_j, which is encrypted using the client j's public key. Again, the proxy $\mathcal{P}$ cannot extract the decryption key d_j.

[1]In practice, the server $\mathcal{S}$ can pre-generate a set of encryption keys $\{e_i\}$ and decryption keys $\{d_i\}$ per multimedia object for each of its authorized clients.

(3) **Decryption key retrieval:** The proxy $\mathcal{P}$ replies back to the client j with the encrypted decryption key. The client j decrypts using its own private key to retrieve the decryption key d_j.
(4) **Data re-encryption and streaming:** The proxy $\mathcal{P}$ uses the re-encryption key e_j and n to re-encrypt the cached multimedia data packets based on the previous ECP setting. It then streams the re-encrypted data packets to the client j via an ordinary and possibly insecure channel. Upon receiving the re-encrypted data packets, the client j can use the decryption key d_j to decrypt the received multimedia data packets.

C. *Encryption Configuration Parameters (ECP)*

Different from text documents, multimedia objects do not require full encryption for proper protection. For example, an MPEG-1 stream consists of three types of frames, namely I frames (interpolative), P frames (predictive), and B frames (bidirectionally predictive). An I frame is typically inserted for every 12 to 15 frames, and operations such as playback, forward, and rewind can only start at I frames. Thus, encryption on all the I frames would provide enough protection but only require about 0.8% of the encryption operations compared to full encryption. Although some I-blocks in the P frames and B frames may still be visible, our goal is to protect the video, when viewed continuously, with maximum encryption efficiency. Protection of some individual video frames may not be guaranteed. Privacy problems may also arise with partial encryption. For example, it may be possible to determine which client is viewing which movie or if two clients are viewing the same movie. We only address the security of the video contents in this paper.

Qualitatively, the degree of partial encryption affects the "choppiness" of the encrypted video playback without a proper decryption key. In applications such as video-on-demand and online-lectures, system throughput or performance is highly important, and encryption security can be viewed as a trade-off against performance. In this paper, we propose to use a *general* encryption method that can reduce the encryption overhead at the server/proxy and the decryption overhead at the end clients, for a *variety* of commonly used video encoding formats (such as MPEG-1, MPEG-2, MP3 and Quicktime). We exploit the observation that, for video encoding that accounts for inter-frame data dependencies, a video stream only needs to be encrypted up to a certain percentage for decoding to be practically useless by an unauthorized viewer – i.e., either the video cannot be decoded, or the quality of the decoded video will be so poor that it is unacceptable for video playback. In general, ECP specifies a packet based encryption pattern given by four adjustable parameters, namely

- S_{pkt}, the expected number of bytes in a data packet.
- E_i – the multimedia stream is to be partitioned into successive groups each having E_i consecutive packets, and a *single* packet encryption operation is to be applied to the first packet of each group.
- E_p – for the packet in which the packet encryption operation is to be applied, E_p specifies the fraction of data within the packet that should be encrypted.
- E_b – for the packet in which the encryption operation is to be applied, E_b specifies the number of encryption blocks that should be evenly distributed within that encryption packet.

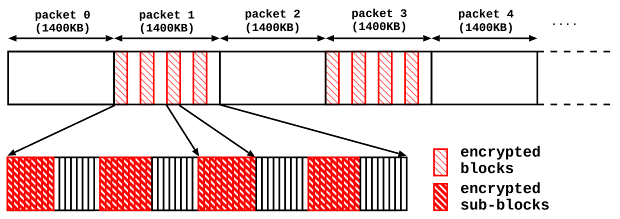


Fig. 4. ECP with $S_{pkt} = 1400$, $E_i = 2$, $E_p = 0.5$, and $E_b = 4$.

In our current implementation, we use UDP as the transport protocol for video data transmission. The entire multimedia stream will be divided into UDP packets with each packet having a payload size of $S_{pkt} = 1400$ bytes. For every $E_i \geq 1$ consecutive UDP packets, we will select the last UDP packet for encryption. For the encrypted packet, it will be further divided into sub-blocks and only some of the sub-blocks will be encrypted. In our current implementation, the sub-block size is chosen to be 4 bytes less than the RSA key length (e.g., 60 bytes for 512-bit RSA) and the encryption will be based on this sub-block unit size. The total length of data to be encrypted within a packet is equal to $E_p \times S_{pkt}$ rounded up to the nearest multiple of the sub-block size. The encrypted sub-blocks will then be regrouped as E_b consecutive blocks of data, and the blocks will be distributed evenly across the whole packet. Once an ECP configuration is selected for a particular video object, the same configuration will be used by the server, the proxy, and the end client in their encryption or decryption operations.

Figure 4 illustrates a possible set of encryption configuration parameters for a multimedia streaming application, where the packet size S_{pkt} is equal to 1400 bytes, $E_i = 2$ (i.e., out of every two consecutive packets, we select the last one for encryption), the fraction E_p is equal to 0.5, and $E_b = 4$ blocks are to be evenly distributed across an encrypted packet. The four configuration parameters allow us to achieve varying degrees of encryption and levels of audio/video quality for the decoded stream. Different video objects can have different characteristics, and thus require different sets of encryption configuration parameters. In our current research, the choice of ECP is by means of experiments. Using our software library, one may try to encrypt with different ECP values and determine a suitable one by observing the visual quality of the encrypted video. Moreover, to prevent any possible attack on multiple streams of the same video object using different ECP values, the ECP configuration should be fixed for any one video object. In Section IV, we illustrate the computational and quality tradeoffs implied by these parameters.

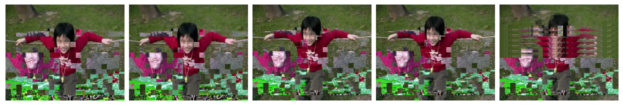
IV. Implementation Results

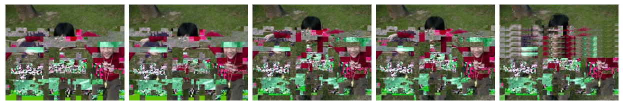
We report experimental results for video streaming using our architecture. We quantify the encryption throughput, the peak signal-to-noise ratio (PSNR), and the related visual quality of video. In our implementation, we use the ECP in Section III-C to control the amount of encryption applied to blocks of audio/video data. The experimental results are taken on an 800 MHz Pentium-III Linux machine with 256 MBytes of main memory. For the experiments, the input data are a set of video sequences, each being an 18 MByte MPEG-1 or a 4.47 MByte Quicktime stream. Because of the lack of space, we refer the reader to [15] for other experiments using MPEG-1/Quicktime video and MP3 audio. Also, please refer to http://www.cse.cuhk.edu.hk/∼cslui/ANSRlab/software/SML/index.html for the Secure Multmedia Library, which is a building block for implementing ARPS-enabled secure multimedia streaming.

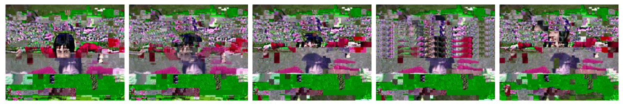
Experiment 1 (Encryption Throughput Analysis): In this experiment, we consider the effect of the parameters E_p and E_i on the encryption throughput, which is denoted as ρ (in MBytes/s). Assume that we are encrypting an MPEG-1 video stream with an average bit rate of 1.5 Mb/s. Given the assumption, the average number of concurrent MPEG-1 streams that a proxy can support is M, where $M = \rho/(1.5/8)$. Table I illustrates the encryption throughput ρ and the average number of concurrent MPEG-1 streams (M) under different values of E_p and E_i, when $E_b = 1$. As we can observe from Table I, if we encrypt 25.7% of *each* video packet (i.e., $E_i = 1$), the encryption throughput achieved is only around 2.13 MBytes/s, which implies that we can only concurrently handle about 11 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 11.82 MBytes/s, which implies that we can concurrently support about 63 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 II illustrates the effect of 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.

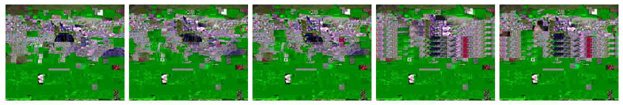
Experiment 2 (Peak Signal-to-Noise Analysis): In this section, we consider the effect on the video quality as we vary the parameters E_i, E_p, and E_b. One way to quantitatively evaluate the video quality is by the peak signal-to-noise ratio. In general, for a frame size of $m \times n$ with a total of l frames and 3 color channels (i.e., red, green, and blue, each represented by an 8-bit number), Note that a lower value of PSNR indicates that the encrypted stream is more distorted from the original video stream. Table III and Table IV illustrate the PSNR for different values of E_p and E_i with $E_b = 1$ for MPEG-1 and Quicktime video, respectively. Note that even when we encrypt one out of 10 video packets, and for a selected packet, we only encrypt 4.3% of the data, we can still obtain a very low value of PSNR. 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.

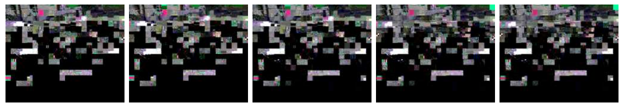
(a) Original frames


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


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


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


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


(f) Encrypted frames:$E_i = 1$, $E_p = 0.043$,$E_b = 1$
(encrypted data are filled with zeros.)

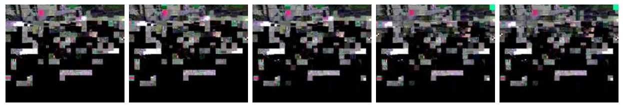

(g) Encrypted frames: $E_i = 1$, $E_p = 0.043$, $E_b = 1$
(encrypted data are being dropped.)

Fig. 5. Quality of five consecutive MPEG-1 video frames under different ECP parameters.

Experiment 3 (Comparison of visual quality of encrypted video): In this experiment, we consider the effect of the ECP parameters E_i, E_p and E_b on the *visual quality* of the video. Figure 5 illustrates the quality of five consecutive MPEG-1 video frames. Figure 5(a) is the original video frames that a client can decode given access to the decryption key. Figures 5(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 $E_i = 1$ and $E_p = 0.043$, which corresponds to encrypting 4.3% of the data for every video packet. Note that when we select $E_i = 10$, $E_p = 0.043$, and $E_b = 1$ (this corresponds to Figure 5(e)), 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

	$\mathbf{E_p = 0.257}$		$\mathbf{E_p = 0.214}$		$\mathbf{E_p = 0.171}$		$\mathbf{E_p = 0.120}$		$\mathbf{E_p = 0.086}$		$\mathbf{E_p = 0.043}$	
	ρ	$\mathbf{M}$	ρ	$\mathbf{M}$	ρ	$\mathbf{M}$	ρ	$\mathbf{M}$	ρ	$\mathbf{M}$	ρ	$\mathbf{M}$
$\mathbf{E_i = 1}$	2.13	11.36	2.53	13.50	3.11	16.60	4.05	21.60	5.8	30.90	10.10	53.90
$\mathbf{E_i = 2}$	4.10	21.87	4.84	25.81	5.91	32.52	7.54	40.20	10.16	54.19	11.77	62.77
$\mathbf{E_i = 5}$	9.06	48.32	10.17	54.24	11.56	61.65	11.64	62.08	11.76	62.72	11.78	62.82
$\mathbf{E_i = 10}$	11.64	62.08	10.70	57.10	11.70	62.40	11.73	62.56	11.73	62.56	11.82	63.04

TABLE I

EFFECT OF E_p AND E_i ON THE ENCRYPTION THROUGHPUT ρ (IN UNIT OF MBYTES/S) AND THE AVERAGE NUMBER OF MPEG-1 STREAMS M WHEN $E_b = 1$.

	encryption throughput ρ (MB/sec)			
	$\mathbf{E_i = 1}$	$\mathbf{E_i = 2}$	$\mathbf{E_i = 5}$	$\mathbf{E_i = 10}$
$\mathbf{E_b = 1}$	2.13	4.10	9.06	11.64
$\mathbf{E_b = 2}$	2.12	4.09	9.01	11.66
$\mathbf{E_b = 3}$	2.12	4.09	9.07	11.65

(a) $E_p = 0.257$

	encryption throughput ρ (MB/sec)			
	$\mathbf{E_i = 1}$	$\mathbf{E_i = 2}$	$\mathbf{E_i = 5}$	$\mathbf{E_i = 10}$
$\mathbf{E_b = 1}$	3.11	5.91	11.56	11.70
$\mathbf{E_b = 2}$	3.11	5.89	11.67	11.72
$\mathbf{E_b = 4}$	3.11	5.89	11.60	11.72

(b) $E_p = 0.171$

TABLE II

EFFECT OF E_i AND E_b ON THE ENCRYPTION THROUGHPUT ρ (IN MBYTES/S) FOR (A) $E_p = 0.257$ AND (B) $E_p = 0.171$.

	peak signal-to-noise ratio (PSNR)					
	$\mathbf{E_p = 0.257}$	$\mathbf{E_p = 0.214}$	$\mathbf{E_p = 0.171}$	$\mathbf{E_p = 0.120}$	$\mathbf{E_p = 0.086}$	$\mathbf{E_p = 0.043}$
$\mathbf{E_i = 1}$	7.83	8.01	8.52	9.32	9.39	8.85
$\mathbf{E_i = 2}$	9.13	8.30	8.70	9.48	9.87	9.51
$\mathbf{E_i = 5}$	11.17	9.81	10.73	10.81	11.39	11.33
$\mathbf{E_i = 10}$	13.06	11.26	12.87	12.60	13.26	12.82

TABLE III

EFFECT OF E_p AND E_i ON THE PEAK SIGNAL-TO-NOISE RATIO (PSNR) ON MPEG-1 VIDEO WHEN $E_b = 1$.

	peak signal-to-noise ratio (PSNR)					
	$\mathbf{E_p = 0.257}$	$\mathbf{E_p = 0.214}$	$\mathbf{E_p = 0.171}$	$\mathbf{E_p = 0.120}$	$\mathbf{E_p = 0.086}$	$\mathbf{E_p = 0.043}$
$\mathbf{E_i = 1}$	11.38	11.56	11.72	12.13	12.28	12.48
$\mathbf{E_i = 2}$	12.15	12.13	12.27	12.55	12.48	12.77
$\mathbf{E_i = 5}$	12.63	12.48	12.57	12.97	12.84	12.98
$\mathbf{E_i = 10}$	12.97	12.76	12.84	13.24	12.98	13.09

TABLE IV

EFFECT OF E_p AND E_i ON THE PEAK SIGNAL-TO-NOISE RATIO (PSNR) ON QUICKTIME VIDEO WHEN $E_b = 1$.

MBytes/s or about 63 concurrent MPEG-1 streams from Table I) and, at the same time, ensure that those clients which do not possess the decryption keys will get unacceptable video quality on viewing. Figure 5 (f)-(g) illustrate the visual effect when attackers intentionally fill encrypted data with zeros or drop the encrypted packets. Again, the result shows that unauthorized access will have poor visual quality.

V. RELATED WORK

Recent research on video proxies has mainly focused on caching strategies and replacement algorithms. Sen and Towsley [11] present how *prefix caching* at a proxy can help to shield clients from large start-up delay, low throughput, and high packet loss. Guo *et al.* [4] propose the use of a prefix-caching proxy in conjunction with a periodic broadcasting technique to improve system scalability. Focusing on implementation and protocol issues, Cruber *et al.* [3] show how to realize proxy prefix caching by using the Real-Time Streaming Protocol (RTSP). Rejaie *et al.* [8] present a fine-grained replacement algorithm for a multimedia proxy, which targets layered-encoded streams. Kangasharju *et al.* [6] 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 emphasizing security issues in a video proxy. Griwodz *et al.* [2] 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 a unicast manner. Since the original server must perform data encryption for each client, this is not a scalable solution. Tosun and Feng [14] 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. Furthermore, the need for decryption operations at the proxy results in higher computational overhead. Shi and Bhargava [12] 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 still decrypt the video in a single operation using a composite decryption key. However, VEA is not resilient against plaintext attack. Therefore, adversaries can obtain the VEA secret key with feasible efforts.

VI. Conclusion

We present the design and implementation of a multi-key secure multimedia proxy architecture. Our design is based on the notion of an *asymmetric reversible parametric sequence* (ARPS). We discussed how ARPS can be applied to a general client-proxy-server architecture. To practically achieve the confidentiality properties of ARPS, we have presented a multi-key RSA technique, and proved that the technique realizes an ARPS. In summary, our theoretical results show that the proposed architecture can achieve comprehensive data confidentiality that is *provably resilient* against attacks, given standard computability assumptions. Our implementation results empirically demonstrate how a set of four ECP parameters can trade off encryption throughput against the amount of data to protect, for a number of standard MPEG-1 and Quicktime video sequences. (Results for a number of MP3 audio sequences are available in [15].) 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 access. For example, by using $E_i = 10$ and $E_p = 0.043$, a single Pentium III/800 MHz machine can concurrently sustain more than 64 *distinct* MPEG-1 video streams, while giving good protection for the original video data. We believe that the proposed system offers an effective approach for delivering multimedia contents in a secure manner.

Acknowledgement:

J. C. S. Lui was supported in part by an RGC Earmarked Grant. D. K. Y. Yau was supported in part by the NSF under grant number CCR-9875742 (CAREER), and in part by CERIAS.

References

[1] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *A. D. Santis (Ed.), Advances in Cryptology: Proceedings of Eurocrypt'94, Vol. 950 of Lecture Notes in Computer Science, Springer-Verlag, 1995.*, pages 92–111, 1995.

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

[3] S. Gruber, J. Rexford, and A. Basso. Protocol considerations for a prefix-caching proxy for multimedia streams. In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, Netherlands, May 2000.

[4] Y. Guo, S. Sen, and D. Towsley. Prefix caching assisted periodic broadcast: Framework and techniques to support streaming for popular videos. In *IEEE ICC*, 2002.

[5] R. S. Inc. Pkcs-1 v2.1. In *RSA Cryptography Standard. Technical Report*, 1999.

[6] J. Kangasharju, F. Hartanto, M. Reisslein, and K. W. Ross. Distributing layered encoded video through caches. In *Proceedings of IEEE Infocom 2001*, pages 1791–1800, Anchorage, Alaska, April 2001.

[7] R. Molva and A. Pannetrat. Scalable multicast security in dynamic groups. In *Proceeding of the 6th ACM Conference on Computer and Communications Security*, pages 101–111, November 1999.

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

[9] B. Schneier. *Applied Cryptography*. John Wiley and Sons, New York, 1996.

[10] R. Security. *PKCS #1: RSA Cryptography Standard.* http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1, 2002.

[11] S. Sen and D. Towsley. Proxy prefix caching for multimedia streams. In *IEEE INFOCOM*, March 1999.

[12] C. Shi and B. Bhargava. A fast MPEG video encryption algorithm. In *Proceeding of the 6th ACM International Multimedia Conference*, pages 81–88, September 1998.

[13] D. R. Stinson. *Cryptography: Theory and Practice (Discrete Mathematics and Its Applications)*. Chapman & Hall; 1st edition, 1995.

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

[15] S. F. Yeung, J. C. S. Lui, and D. K. Y. Yau. A multi-key secure multimedia proxy using asymmetric reversible parametric sequences: Theory, design, and implementation. Technical report, Chinese University of Hong Kong, 2003. Also available as CS-TR-LANS-03-2, Purdue University, West Lafayette, IN.

S. F. Yeung was born in Hong Kong. He received his B.Eng and M.Phil. in Computer Science from the Chinese University of Hong Kong (CUHK). He is presently the CTO of G&B Studio. His research interests 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 Jose Laboratory 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, distributed 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 of the 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.

David K. Y. Yau received the B.Sc. (first class honors) degree from the Chinese University of Hong Kong, and the M.S. and Ph.D. degrees from the University of Texas at Austin, all in computer sciences. From 1989 to 1990, he was with the Systems and Technology group of Citibank, NA. He was the recipient of an IBM graduate fellowship, and is currently an Associate Professor of Computer Sciences at Purdue University, West Lafayette, IN. He received an NSF CAREER award in 1999, for research on network and operating system architectures and algorithms for quality of service provisioning. His other research interests are in network security, value-added services routers, and mobile wireless networking.