A Case for a Multi-Key Secure Video Proxy: Theory, Design, and Implementation

Siu F. Yeung
Department of Computer Science & Engineering
The Chinese University of Hong Kong
sfyeung@cse.cuhk.edu.hk

John C. S. Lui*
Department of Computer Science & Engineering
The Chinese University of Hong Kong
cslui@cse.cuhk.edu.hk

David K. Y. Yau†
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
yau@cs.purdue.edu

ABSTRACT

Because of limited server and network capacities in multimedia streaming, *proxies* are commonly used to cache multimedia objects such that, by accessing nearby proxies, clients can enjoy smaller start-up latencies and reduced packet loss and delay jitters for their requests. However, the use of video 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 video proxy or, more generally, a secure proxy architecture. The framework employs a notion of *asymmetric reversible parametric sequences* to provide the following security 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. Our framework is grounded on a *multi-key RSA* technique such that system resilience against attacks is provably strong given standard computability assumptions. We also propose the use of a set of *encryption configuration parameters* to trade off proxy encryption throughput against the viewing quality of video by unauthorized parties. Implementation results on a Pentium III/800 MHz machine show that our techniques can *simultaneously* achieve high encryption throughput and extremely low video quality (in terms of both PSNR and the visual quality of decoded frames) during unauthorized viewing.

Keywords

Security, Asymmetric Parametric Sequence Functions, Multi-Key RSA, Video Proxy

*Research supported in part by the AOE and RGC Research Grant.

†Research supported in part by the National Science Foundation under grant numbers CCR-9875742 (CAREER) and EIA-9806741.

1. INTRODUCTION

The emergence of the Internet as a pervasive form of communication, as well as advances in digital video and compression technologies, has led to the recent wide deployment of continuous media streaming over the network. A wide range of applications such as video-on-demand, distance learning, and corporate telecasts and narrowcasts are now enabled by the ability to stream video/audio data from servers to clients across a wide area. However, because of the high bandwidth requirement (e.g., a high quality video stream usually has a bandwidth requirement of over 1 Mb/s) and the long duration nature (e.g., from tens of minutes to several hours) of digital video, server and network bandwidths are major limiting factors in achieving a *scalable* streaming service. Consequently, there has been a lot of research on developing techniques for bandwidth-efficient distribution of multimedia data to a large client population. One common solution, for example, is to use a multimedia proxy to perform some form of data caching (say, prefix caching), so that clients can access the cached video from their nearby proxies to minimize delay and conserve bandwidth.

One major problem with the multimedia proxy approach is the risk of revealing the original video data to unauthorized parties. For example, when the original data are sent from server to proxy, anyone that eavesdrops on the communication link between the source and the proxy can gain access to the video information. Some "naive" approaches to counter the problem are:

- **Encryption using a secret key between the server and the proxy:** Under this approach, the server and the proxy will exchange a secret key $\mathcal{X}$ for encrypting the video data. The source encrypts the data based on $\mathcal{X}$ and sends the encrypted data to the proxy. The proxy, upon receiving the encrypted data, can perform decryption and cache the video data in clear form. There are several problems with this approach, including:
  - *Data insecurity at the proxy:* Since the cached data at the proxy are the original multimedia data, any intruder who gains access to the proxy's storage can access the original data.
  - *Data insecurity between the proxy and clients:* Since

the data transfer between the proxy and its clients can be over an insecure channel, one can eavesdrop on this channel and gain access to the original multimedia data.

- **End-to-end encryption using a secret key between the server, the proxy and the clients:** Under this approach, the server, the proxy and all the clients behind the multimedia proxy will share a common secret key $\mathcal{X}$. The source encrypts the data based on $\mathcal{X}$ and sends the data to the multimedia proxy. The proxy, upon receiving the encrypted data, caches the data in its local storage. Whenever a client wants to access the multimedia data, the encrypted copy will be sent to that client. Since the client also knows the secret key, it can decrypt and extract the original data. An intruder can still eavesdrop on the communication link between the proxy and the client, but it will not be able to decrypt the data. The major problem with this approach is that there is a high risk of revealing the secret key $\mathcal{X}$. The reason is that a proxy needs to support a large number of clients, and if *any* of these clients is compromised, an intruder can use the revealed secret key $\mathcal{X}$ to gain access to the original multimedia data.

- **Heterogeneous secret keys between the server-proxy pair and the proxy-client pairs:** Under this approach, the server encrypts the data based on a common secret key $\mathcal{X}$ that is shared between the server and the proxy only. Upon receiving the data, the proxy caches the encrypted data. Whenever a client, say i, wants to access the multimedia data, the proxy will (1) decrypt the data based on the secret key $\mathcal{X}$, (2) encrypt the data based on a common secret key $\mathcal{X}_i$ between the proxy and the client i. The client i, upon receiving the encrypted data, can gain access to the original data because it knows the common secret key $\mathcal{X}_i$. The potential problem with this approach is that it requires high computational overhead at the proxy, because the proxy needs to perform decryption and encryption for every admitted client. This can limit the number of concurrent clients that the proxy can support, and if the proxy is compromised, an intruder can reveal the secret key $\mathcal{X}$.

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

- The multimedia proxy will only cache *encrypted* video data and data decryption will only happen at the end-points (e.g, the clients). Therefore, the original video data will not be revealed at any intermediate node.

- The multimedia proxy will perform encryption operations only (i.e., it performs no decryption at all); this reduces the computational overhead at the proxy, and hence allows to support a higher number of concurrent clients. Also, even when the proxy is compromised, an intruder cannot reveal any useful information.

- Data encryption and decryption operations are based on well accepted encryption theory that it is computationally infeasible to extract the original multimedia data without knowledge of the expected decryption key.

- *Membership collusion* is 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, one still cannot derive the original video data.

The rest of the paper is organized as follows. In Section 2, we present multi-key encryption based on the *asymmetric reversible parametric sequence.* We then present an practical algorithm to implement the asymmetric reversible parametric sequences such that the proxy server will have the claimed properties. In Section 3, we present the architecture of our video proxy and the communication protocols between (1) the proxy and the server, and (2) the clients and the proxy. In Section 4, we report experiments to illustrate the encryption data rate achievable on a commodity Pentium machine, and give quantitative and qualitative analysis of the encrypted video quality. Related work in multimedia proxies is presented in Section 5. Section 6 concludes.

2. 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) [5]. We first present the formal definition of RPS and its utilities. We then present an implementation of RPS using the multi-key RSA technique.

2.1 Reversible Parametric Sequence

Let $f : I\!N^2 \rightarrow I\!N$ 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. (Elements in this sequence do not need to be unique.) Define a finite data transformation sequence

$$\mathcal{D} = \{D_{-1}, D_0, ..., D_N\}$$

based on the function f and the finite sequence $\{e_i\}_{0 \leq i \leq N}$ such that

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

We have the following definitions of *reversible parametric sequences.*

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 $-1 \leq i < j \leq N$, there exists a* computable *function $\Omega_{i,j}$ such that*

$$D_i = \Omega_{i,j}(D_j) \quad \text{for } -1 \leq i < j \leq N.$$

Definition 2. *A 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 $\{e_{i+1}, \cdots, e_j\}$.*

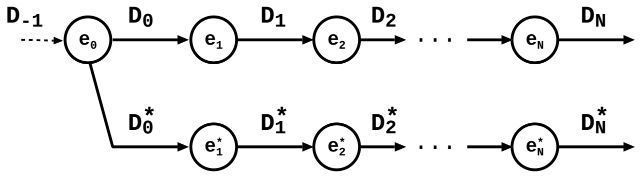


Figure 1: A graphical representation of two RPS_f sequences.

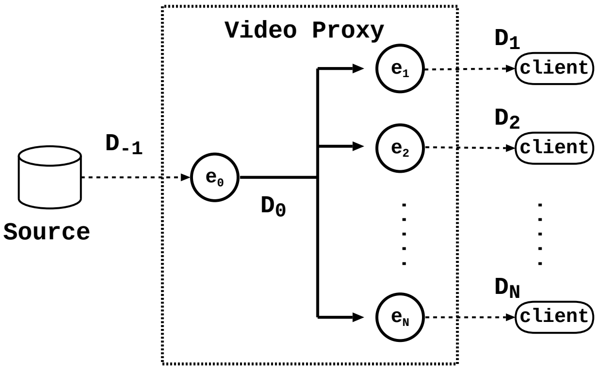


Figure 2: A graphical representation of an $ARPS_f$ sequence for the secure video proxy.

Definition 3. *A 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 $\{e_{i+1}, \cdots, e_j\}$.*

To understand this concept, 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 (or D_0^*) using $D_0 = f(D, e_0)$. If $e_i \neq e_i^*$, then the intermediate data D_i will not be equal to D_i^*, for $1 \leq i \leq N$. 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 $0 \leq j \leq N$. 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* compute the original data D_{-1} even if given the data D_j (or D_j^*) and the sequence $\{e_0, \cdots, e_j\}$ (or $\{e_0^*, \cdots, e_j^*\}$), for $0 \leq j \leq N$.

One can use the properties of an asymmetric reversible parametric sequence $ARPS_f$ to implement a secure video proxy. To illustrate, consider a graphical representation of an $ARPS_f$ sequence in Figure 2. The proxy can request to obtain 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 D_{-1}, if i is given a decryption function $\Omega_{-1,j}$ (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 $0 \leq i \leq N$, it still cannot decrypt any of the encrypted data D_i being cached, for $0 \leq i \leq N$, in order to obtain the original data D_{-1}.

Therefore, one can use an asymmetric reversible parametric sequence $ARPS_f$ to implement a video proxy which has the following properties:

- **Data confidentiality during transmission:** Since the original data D_{-1} are encrypted, an intruder who eavesdrops on the link between the source and the proxy can only access D_0 and will not be able to extract the original data. The same conclusion holds when an intruder eavesdrops on the link between the proxy and the client i. The intruder can only access D_i and will not be able to extract the original data.

- **End-to-end data confidentiality:** Because of the RPS_f, decryption of the original data D_{-1} only takes place at the endpoints (e.g., the clients). The proxy and clients only store the data encrypted as D_0 and D_i. Hence, even if an intruder gains access to the proxy's or client's local storage, the original data D_{-1} will not be revealed.

- **Data confidentiality against proxy intruders:** If an intruder compromises the proxy server or a client's machine, and if the encryption process is an $SRPS_f$, then the intruder, on knowing e_0 and e_i, can gain access to D_{-1}. This is because the function $\Omega_{-1,0}$ can be computed from the knowledge of e_0 and e_i such that the original data can be revealed by $D_{-1} = \Omega_{-1,0}(D_0)$. On the other hand, if the encryption process is an $ARPS_f$, then even if e_0 and e_i are compromised, the original data D_{-1} cannot be revealed, because the function $\Omega_{-1,0}$ is computationally infeasible to determine in this case.

- **Data confidentiality against member collusion:** If the encryption process is $SRPS_f$, member collusion is possible when a client j gains access to 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, given e_i, the intruder can obtain $\Omega_{0,i}$ and thus obtain $D_0 = \Omega_{0,i}(D_i)$. Given the knowledge of e_j and D_0, it can obtain D_j by $D_j = f(D_0, e_j)$. 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.

2.2 Implementation of $ARPS_f$

We need a practical and efficient algorithm to realize the properties of an asymmetric reversible parametric sequence. In our work, we use *multi-key RSA* to implement an $ARPS_f$. We first present the basic idea of RSA. We then extend the concept to a multi-key RSA framework.

For standard RSA (or *single-key* RSA), one needs to generate two distinct large prime number p and q. Let us define

$$\begin{aligned} n &= pq, \\ \phi &= (p-1)(q-1). \end{aligned}$$

To encrypt a given data item D_{-1}, one has to find an encryption key e_0 such that the integer e_0 satisfies

- $1 < e_0 < \phi$, and
- $gcd(e_0, \phi) = 1$.

To encrypt the data D_{-1}, we generate a cipher C based on the encryption key e_0 wherein

$$C = (D_{-1})^{e_0} (\text{mod } n).$$

The cipher C can be transmitted over an insecure channel. The receiver needs to have a decryption key d_0 to decrypt the cipher. This decryption key d_0 is an integer and is selected such that:

- $1 < d_0 < \phi$, and
- $(e_0)d_0 = 1 (\text{mod } \phi)$.

Upon receiving the cipher, the receiver can decrypt the cipher C and obtain the original data D_{-1} by

$$D_{-1} = C^{d_0} (\text{mod } n).$$

We can extend the single-key RSA technique to a *multi-key* RSA technique. Consider the proxy server as an example. The proxy server needs to generate two large prime numbers, say p and q. In addition, it needs to generate a sequence of encryption keys $\{e_i\}_{0 \leq i \leq N}$ such that

$$1 < e_i < \phi \qquad \text{and} \qquad (1)$$
$$gcd(e_i, \phi) = 1. \qquad (2)$$

The proxy server will send the encryption key e_0 and n over a secure channel to the source. The source will encrypt the original data D_{-1} using e_0 and generate a cipher D_0 using

$$D_0 = (D_{-1})^{e_0} (\text{mod } n). \qquad (3)$$

Upon receiving the cipher D_0, the proxy can store the encrypted data in its local storage. Whenever a client i wants 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

$$D_i = (D_0)^{e_i} (\text{mod } n). \qquad (4)$$

Moreover, the proxy generates a decryption key d_i for client i. The decryption key d_i has to satisfy the following two conditions:

$$1 < d_i < \phi \qquad \text{and} \qquad (5)$$
$$(e_0 e_i) d_i = 1 (\text{mod } \phi). \qquad (6)$$

Computing these decryption keys d_i can be easily achieved using the Extended Euclidean Algorithm [7]. The proxy sends 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:

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

Example: To illustrate, consider the following simple example. Suppose that the two primes are $p = 5$ and $q = 7$ (in reality, p and q have to be large). Accordingly, $n = 5 \times 7 = 35$ and $\phi = (5-1) \times (7-1) = 24$. Let there be three encryption keys: $e_0 = 5$, $e_1 = 11$ and $e_2 = 13$. If the original data $D_{-1} = 10$, the cached data in the proxy server will be

$$D_0 = 10^5 (\text{mod } 35) = 5.$$

When client 1 requests the data, the proxy will generate a decryption key d_1 such that $(5 \times 11)d_1 = 1 (\text{mod } 24)$. Using the Extended Euclidean Algorithm, we have $d_1 = 7$. Therefore, the encrypted data for client 1 is

$$D_1 = (D_0)^{e_1} (\text{mod } n) = 5^{11} (\text{mod } 35) = 10$$

and client 1 can decrypt the data D_1 and get back the original data D_{-1} by

$$D_{-1} = (D_1)^{d_1} (\text{mod } n) = 10^7 (\text{mod } 35) = 10.$$

When client 2 requests the data, the proxy will generate a decryption key d_2 such that $(5 \times 13)d_2 = 1 (\text{mod } 24)$. Using the Extended Euclidean Algorithm, we have $d_2 = 17$. Therefore, the encrypted data for client 1 is

$$D_2 = (D_0)^{e_2} (\text{mod } n) = 5^{13} (\text{mod } 35) = 5$$

and client 2 can decrypt the data D_2 and get back the original data D_{-1} by

$$D_{-1} = (D_2)^{d_2} (\text{mod } n) = 5^{17} (\text{mod } 35) = 10.$$

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

Proof: Please refer to [11] for a proof. ∎

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

Proof: Please refer to [11] for a proof. ∎

3. VIDEO PROXY: ARCHITECTURES AND PROTOCOLS

In this section, we describe in detail our server-proxy-client architecture. The video server, the video proxy and the clients have been implemented in C++ on a Linux platform. In our current implementation, we use a 512-bit RSA key to implement various encryption and decryption operations, but this can be easily extended to accommodate RSA keys with a larger number of bits. Security features such as keys 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. GMP 4.0 provides one of the fastest possible arithmetic library for applications that need higher precision than is directly supported by the basic C types.

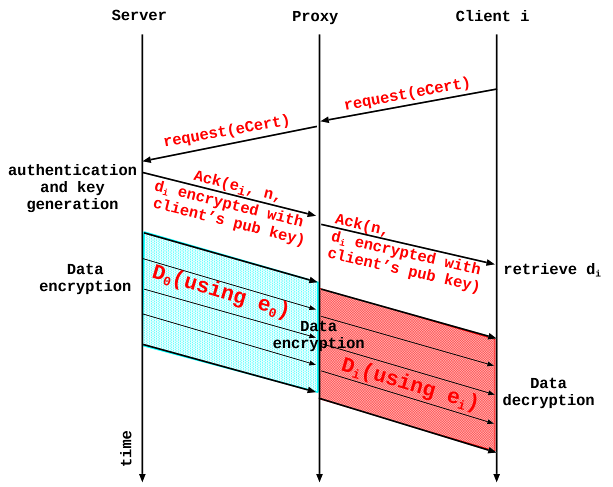


Figure 3: Operations between the source video server $\mathcal{S}$ and the proxy $\mathcal{P}$, and operations between the proxy $\mathcal{P}$ and the client i.

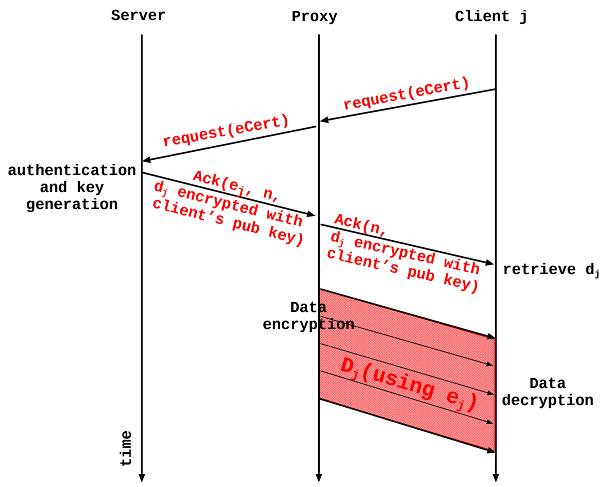


Figure 4: Operations between the source video server $\mathcal{S}$ and the proxy $\mathcal{P}$, and operations between the proxy $\mathcal{P}$ and the client i.

3.1 Operations to Request and Cache Data from Server

Figure 3 illustrates the operations between the video 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 video data. These operations are:

1. **Initiate connection:** The client i sends a video request and its certificate to the proxy $\mathcal{P}$, the proxy $\mathcal{P}$ forwards the request to the video server $\mathcal{S}$. The video server $\mathcal{S}$ needs to authenticate that the request is indeed from the client i. If the authentication succeeds, the server $\mathcal{S}$ will go on to the key generation operation.

2. **Key generation:** The server randomly generates two large prime numbers p and q, computes the prime product $n = pq$, the pseudo-prime product $\phi = (n-1)(p-1)$, the encryption key e_0, re-encryption key e_i and the corresponding decryption key d_i via Equations (1) and (2). The server $\mathcal{S}$ saves the parameters ϕ, e_0 and n with a unique identifier ID. It then replies an acknowledgment back to the proxy $\mathcal{P}$ with the re-encryption key e_i, and the corresponding decryption key d_i which encrypted using the client i's public key.

3. **Decryption key retrieval:** The proxy $\mathcal{P}$ replies with an acknowledgment back to the client i with the encrypted decryption key. The client i decrypts using its own private key to retrieve the decryption key d_i.

4. **Data encryption and streaming:** The video server $\mathcal{S}$ uses the encryption key e_0 and n to encrypt the video data packets. The degree of encryption is based on the encryption configuration parameters (ECP) which we will describe in Section 3.3. The video 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.

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 video data packets in the cache. The encryption is based on the same ECP setting as the server $\mathcal{S}$ used. 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 decrypts the data using the decryption key d_i.

3.2 Operations to Request Cached Data from Proxy

Figure 4 illustrates the operations between the video server $\mathcal{S}$ and the proxy $\mathcal{P}$, and the operations between the proxy $\mathcal{P}$ and the client i for requesting a video data that is 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 specifying the unique identifier ID. The video 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 replies the proxy $\mathcal{P}$ an acknowledgment with the re-encryption key e_j, and the decryption key d_j which encrypted using the client j's public key.

3. **Decryption key retrieval:** The proxy $\mathcal{P}$ replies with an acknowledgment 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 video 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 decrypts the data using the decryption key d_j.

3.3 Encryption Configuration Parameters

Although the proxy $\mathcal{P}$ does not need to perform any decryption, the RSA encryption operation – even when implemented using the optimized GMP 4.0 library – can still be expensive. Since $\mathcal{P}$ has to perform encryption for every client i, the encryption overhead can limit the achievable streaming throughput and the number of concurrent clients that the proxy can support.

To reduce the computational overhead at the proxy we perform encryption based on specifiable *encryption configuration parameters* (ECP). The idea consists in not performing encryption on an entire video stream, but instead on selected data blocks in the video stream. Note that for certain video formats, encrypting a complete video stream is inherently redundant. For example, if video data are encoded using MPEG-1, then one might argue that encrypting all the I frames in the video sequence would suffice, since the other frames depend on the I frames in decoding.

We propose to use a *general* encryption method that can reduce the encryption overhead at the proxy for a *variety* of commonly used video encoding formats (such as MPEG-1, MPEG-2, 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, in that either the video cannot be decoded, or the quality of the decoded video will be so poor that it is unacceptable for viewing. ECP specifies a packet based encryption pattern given by four adjustable parameters, namely

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

In our current implementation, we use UDP as the transport protocol for data transmission. The entire video stream will be divided into UDP packets with each packet having a payload size of $S_{pkt} = 1400$ bytes. For every $I \geq 1$ consecutive UDP packets, we will select the last UDP packet for encryption. For the encrypted packet, it will be further divided up 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 $P * S_{pkt}$ rounded up to the nearest multiple of the sub-block size. The encrypted sub-blocks will then be regrouped as B consecutive blocks of data, and and the blocks will be distributed evenly across the whole packet.

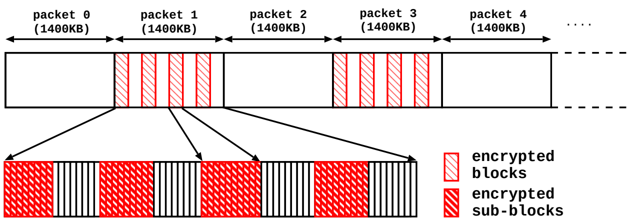


Figure 5: Illustration of ECP with $S_{pkt} = 1400$, $I = 2$, $P = 0.5$, and $B = 4$.

Figure 5 illustrates a possible set of encryption configuration parameters, where the packet size S_{pkt} is equal to 1400 bytes, $I = 2$ (i.e., out of every two consecutive packets, we select the first one for encryption), the fraction P is equal to 0.5, and and $B = 4$ 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 video quality for the decoded stream. In Section 4, we illustrate the computational and quality tradeoffs implied by these parameters.

4. EXPERIMENTS

In this section, we report experiments that quantify the encryption throughput, and the peak signal-to-noise ratio (PSNR) and the visual quality of the decoded video, in the context of our secure video distribution architecture.

In our current implementation, we use encryption configuration parameters (ECP) to control the amount of encryption applied to blocks of video data. The experimental results are taken on an 800 MHz Pentium-III Linux machine with 256 MBytes of main memory. The video samples used in these experiments are a set of video sequences, each being an 18 MByte MPEG-1 stream or a 4.47 MByte Quicktime stream.

Experiment 1: Encryption Throughput Analysis: In this experiment, we consider the effect of the parameters P and I on the encryption throughput, measured as ρ in MBytes/s. Assume that we are encrypting an MPEG-1 stream with an average bit rate of 1.5 Mbps. 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 1 illustrates the encryption throughput ρ and the average number of concurrent MPEG-1 streams (M) under different values of P and I, when $B = 1$. As we can observe from Table 1, if we encrypt 25.7% of *each* video packet (i.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 pack-

	$P = 0.257$		$P = 0.214$		$P = 0.171$		$P = 0.120$		$P = 0.086$		$P = 0.043$	
	ρ	M	ρ	M	ρ	M	ρ	M	ρ	M	ρ	M
$I = 1$	2.13	11.36	2.53	13.50	3.11	16.60	4.05	21.60	5.8	30.90	10.10	53.90
$I = 2$	4.10	21.87	4.84	25.81	5.91	32.52	7.54	40.20	10.16	54.19	11.77	62.77
$I = 5$	9.06	48.32	10.17	54.24	11.56	61.65	11.64	62.08	11.76	62.72	11.78	62.82
$I = 10$	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 1: Effect of P and I on the encryption throughput ρ (in unit of MBytes/s) and the average number of MPEG-1 streams M when $B = 1$.

ets (i.e., $I = 10$) and for each video packet encrypted, we encrypt only 4.3% of its data, 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 values of P and I, the higher the encryption throughput and the number of concurrent video streams the system can handle.

Table 2 illustrates the effect of I and B under two different encryption percentage parameters P. As we can observe, the parameter 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 I, P and B. One way to quantitatively assess 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 (e.g., 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 $c = 3$ correspond 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 highly distorted from the original video data stream.

Table 3 and Table 4 illustrate the peak signal-to-noise ratio SNR_{peak} for different values of P and I with $B = 1$ 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 still obtain a very low value of SNR_{peak}. This indicates that (1) we can apply this encryption technique for different video formats, and (2) we only need to encrypt a small fraction of the video data to achieve *both* high encryption throughput and high video distortion.

Experiment 3: Comparison of visual quality of encrypted video: In this experiment, we consider the effect of the ECP parameters I, P and B on the *visual quality* of the video. Figure 6 illustrates the quality of five consecutive MPEG-1 video frames. Figure 6(a) is the original video frames that a client can decode given access to the decryption key. Figures 6(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 $I = 1$ and $P = 0.043$, which corresponds to encrypting 4.3% of the data for every video packet. Note that when we select $I = 10$, $P = 0.043$, and $B = 1$ (this corresponds to Figure 6(e)), the visual quality of the video is still quite unacceptable for viewing. This shows that we can achieve high encryption throughput (e.g., 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 unacceptable video quality on viewing. Figure 7 shows the corresponding results for five consecutive Quicktime video frames.

5. RELATED WORK

Recent research on video proxies has mainly focused on caching strategies and replacement algorithms. Sen and Towsley [8] 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.* [3] 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.* [2] show how to realize proxy prefix caching by using the Real-Time Streaming Protocol (RTSP). Rejaie *et al.* [6] present a fine-grained replacement algorithm for a multimedia proxy, which targets layered-encoded streams. Kangasharju *et al.* [4] 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 number of papers emphasizing security issues in a video proxy. Griwodz *et al.* [1] 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 [10] 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

	encryption throughput ρ (MB/sec)			
	$I = 1$	$I = 2$	$I = 5$	$I = 10$
$B = 1$	2.13	4.10	9.06	11.64
$B = 2$	2.12	4.09	9.01	11.66
$B = 3$	2.12	4.09	9.07	11.65

(a) P=0.257

	encryption throughput ρ (MB/sec)			
	$I = 1$	$I = 2$	$I = 5$	$I = 10$
$B = 1$	3.11	5.91	11.56	11.70
$B = 2$	3.11	5.89	11.67	11.72
$B = 4$	3.11	5.89	11.60	11.72

(b) P=0.171

Table 2: Effect of I and B on the encryption throughput ρ (in MBytes/s) for (a) $P = 0.257$ and (b) $P = 0.171$.

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

Table 3: Effect of P and I on the peak signal-to-noise ratio SNR_{peak} on MPEG-1 video when $B = 1$.

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 [9] 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. Hence, while the approach is highly efficient, more determined adversaries can obtain the VEA secret key with feasible efforts.

6. CONCLUSION

We have presented the design and implementation of a multi-key secure video 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 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.

We have an implementation of our video streaming architecture – consisting of server, proxy and client – on commodity Pentium III/800 MHz machines running Linux. Our implementation results empirically demonstrate how a set of four ECP parameters can trade off encryption throughput against data confidentiality, for a number of standard MPEG-1 and Quicktime video sequences. They indicate that it is possible to *simultaneously* achieve high encryption throughput and extremely low video quality (in terms of both PSNR and the visual quality of decoded frames) during unauthorized viewing. For example, by using $I = 10$ and $P = 0.043$ 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.

7. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

[9] 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.

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

[11] S. F. Yeung, J. C. S. Lui, and D. K. Y. Yau. A case for a multi-key secure video proxy: Theory, design, and implementation. Technical report, Dept of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, 2002. Also as CS TR-02-011, Purdue University, West Lafayette, IN.

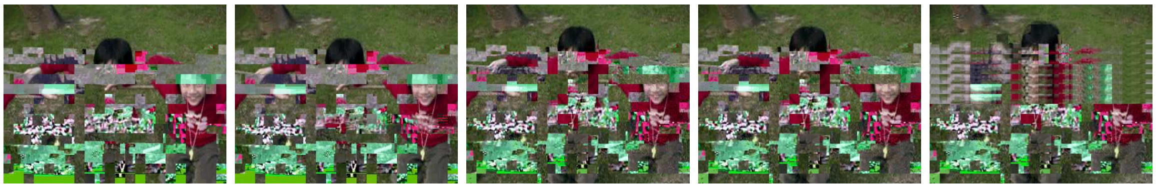
	peak signal-to-noise ratio SNR_{peak}					
	$P = 0.257$	$P = 0.214$	$P = 0.171$	$P = 0.120$	$P = 0.086$	$P = 0.043$
$I = 1$	11.38	11.56	11.72	12.13	12.28	12.48
$I = 2$	12.15	12.13	12.27	12.55	12.48	12.77
$I = 5$	12.63	12.48	12.57	12.97	12.84	12.98
$I = 10$	12.97	12.76	12.84	13.24	12.98	13.09

Table 4: Effect of P and I on the peak signal-to-noise ratio SNR_{peak} on Quicktime video when $B = 1$.

(a) Original frames

(b) Encrypted frames with $I = 1$, $P = 0.043$ and $B = 1$.

(c) Encrypted frames with $I = 2$, $P = 0.043$ and $B = 1$.



(d) Encrypted frames with $I = 5$, $P = 0.043$ and $B = 1$.

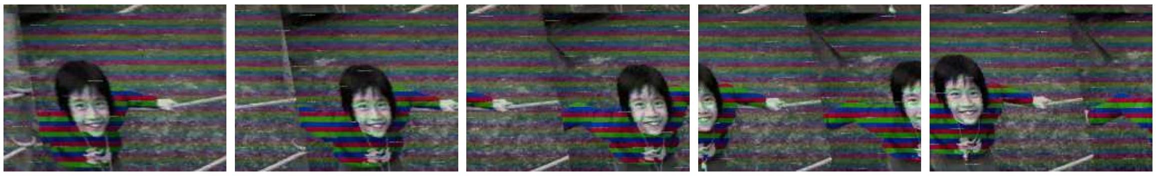
(e) Encrypted frames with $I = 10$, $P = 0.043$ and $B = 1$.

Figure 6: Quality of five consecutive MPEG-1 video frames under different ECP parameters.

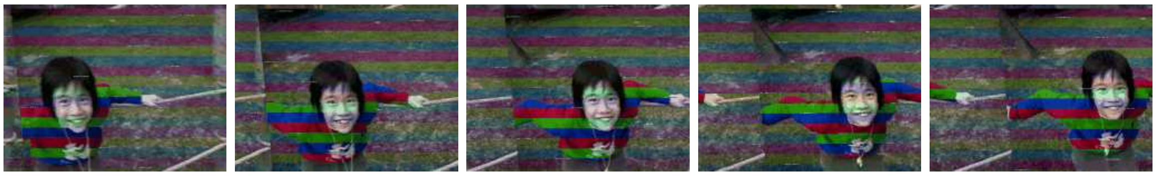
(a) Original frames

(b) Encrypted frames with $I = 1$, $P = 0.043$ and $B = 1$.

(c) Encrypted frames with $I = 2$, $P = 0.043$ and $B = 1$.



(d) Encrypted frames with $I = 5$, $P = 0.043$ and $B = 1$.



(e) Encrypted frames with $I = 10$, $P = 0.043$ and $B = 1$.

Figure 7: Quality of five consecutive Quicktime video frames under different ECP parameters.