Efficient and Secure Content Processing and Distribution by Cooperative Intermediaries

Yunhua Koglin, Danfeng Yao, *Member*, *IEEE*, and Elisa Bertino, *Fellow*, *IEEE*

Abstract—Content services such as content filtering and transcoding adapt contents to meet system requirements, display capacities, or user preferences. Data security in such a framework is an important problem and crucial for many Web applications. In this paper, we propose an approach that addresses data integrity and confidentiality in content adaptation and caching by intermediaries. Our approach permits multiple intermediaries to simultaneously perform content services on different portions of the data. Our protocol supports decentralized proxy and key management and flexible delegation of services. Our experimental results show that our approach is efficient and minimizes the amount of data transmitted across the network.

Index Terms—Data sharing, distributed systems, integrity, security.

1 Introduction

In order to enhance the performance of content distribution networks (CDNs), several approaches have been developed based on the use of *content management services* provided by *intermediary proxies*. In most of these approaches, content caching is the main service provided by proxies [1], [3], [15], [18]. That is, instead of asking a content server for contents upon each client request, a proxy first checks if these contents are locally cached. Only when the requested contents are not cached or out of date are the contents transferred from the content server to the clients. If there is a cache hit, the network bandwidth consumption can be reduced. A cache hit also reduces access latency for the clients. System performance thus improves, especially when a large amount of data is involved. Besides these improvements, caching makes the system robust by letting caching proxies provide content distribution services when the server is not available.

With the emergence of various network appliances and heterogeneous client environments, there are other relevant new requirements for content services by intermediaries [2], [10]. For example, content may be transformed to satisfy the requirements of a client's security policy, device capabilities, preferences, and so forth. Therefore, several *content services* have been identified that include but are not limited to content transcoding [2], [5], [10], [13], in which data is transformed from one format into another, data filtering, and value-added services such as watermarking [7]. Other relevant services are related to personalization, according to which special-purpose proxies can tailor the contents based on user preferences, current activities, and past access history.

Many studies have been carried out on intermediary content services [2], [5], [10], [13]; however, the problem of data security in these settings has not caught much attention. Confidentiality and integrity are two main security properties that must be ensured for data in several distributed cooperative application domains such as collaborative e-commerce [20], distance learning, telemedicine, and e-government. *Confidentiality* means that data can only be accessed under the proper authorizations. *Integrity* means that data can only be modified by authorized subjects. The approaches developed for securely transferring data from a server to clients are not suitable when data is to be transformed by intermediaries. When a proxy mediates data transmission, if the data is enciphered during transmission, security is ensured; however, it is impossible for intermediaries to modify the data. On the other hand, when intermediaries are allowed to modify the data, it is difficult to enforce security.

Much previous work has been done on data adaptation and content delivery. The work by Lum and Lau discussed the trade-off between the transcoding overhead and spatial consumption in content adaptation [16]. CoralCDN, a peer-to-peer CDN, was recently presented; it combines peer-to-peer systems and Web-based content delivery [11]. Chi and Wu [8] proposed a Data Integrity Service Model (DISM) to enforce the integrity of data transformed by intermediaries. In such a model, integrity is enforced by using metadata expressing modification policies specified by content owners. However, in DISM, every subject can access the data. Thus, confidentiality is not enforced. Another problem with DISM is the lack of efficiency. It does not exploit the possible parallelism that is inherent in data relationships and in the access control policies. In several applications such as multimedia content adaptation [2] efficiency is crucial. In the partial and preliminary version of this paper [14], a protocol was proposed to ensure confidentiality and integrity for XML document updates in distributed and cooperative systems. In this paper, we present a general and improved protocol to meet the high availability requirement for large-scale network services [10].

- *Y. Koglin is with Cisco Systems, RCDN6/3/4, 2200 East President George Bush Highway, Richardson, TX 75082-3550. E-mail: ykoglin@cisco.com.*
- *D. Yao is with the Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854-8019. E-mail: danfeng@cs.rutgers.edu.*
- *E. Bertino is with the Department of Computer Science, Purdue University, 305 N. University Street, West Lafayette, IN 47907-2107. E-mail: bertino@cs.purdue.edu.*

Manuscript received 24 Feb. 2006; revised 10 May 2007; accepted 20 June 2007; published online 14 Aug. 2007.
Recommended for acceptance by K. Hwang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0039-0206.
Digital Object Identifier no. 10.1109/TPDS.2007.70758.

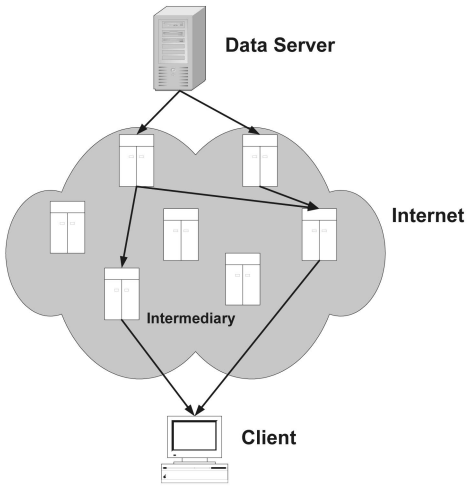

Fig. 1. System architecture.

Our contribution. We summarize our contributions as follows:

1. We describe the security and content transformation involved with cache proxies. We present a parallel secure content service (PSCS) protocol for a cache proxy and analyze the properties of intermediaries with caching capacity.
2. We formalize the key management mechanism in cooperative intermediaries. We introduce the intermediary profile table for the data server to store public keys of peer proxies (P-proxies), which are proxies authorized to perform the same type of data transformation. Our key management does not require any preexisting public key infrastructure. This is possible because the public keys of proxies are endorsed by the data server in the control information. Therefore, public-key certificates are not required in our protocol, even though the proxies do not need to know each other a priori.
3. We implement our protocol and report the experiment results on data size, integrity check time, and servicing time, including the effect of recovery. We also compare and analyze the performance of our protocol with a centralized implementation.
4. We describe and analyze the delegation of authorization among cooperative intermediaries. When an intermediary is overloaded, our approach makes it possible for the intermediary to delegate the execution of content services to another proxy without violating security requirements. Our delegation mechanism is simple to implement, yet it largely improves the availability of proxies.

In our model (see Fig. 1), we distinguish three types of entities:

1. *Data Server.* This is an entity that *originally* stores the data requested by a client.
2. *Client.* This is any entity that requests data from a data server. When a client submits a request, besides the data it requests, it may also include some content service requirements, arising from device limitations and data format limitations [4]. If the client does not specify any service requirements, a proxy that represents the client may add these requirements. Such a proxy may be an edge proxy [5].
3. *Intermediary.* This is any entity that is allowed by a data server to provide content services in response to requests by clients. Intermediaries include caching proxies and transforming proxies.

Function	Purpose	privilege
Customize	Tailor for particular users	Update
Filter	Remove information	Update
Annotate	Add information	Update
Transcode	Change to different format	Update
Cache	Store for later use	Read
Aggregate	Combine from multiple sources	Read
Virus scan	Scan virus	Read, Update
Watermark	Add watermarking	Update

Fig. 2. Functions and corresponding privileges.

Our solution uses standard cryptographic primitives, including a collision-resistant hash function and digital signatures. We also design a data structure, called *control information*, for the data server to manage proxies and authorizations. Each participant (intermediary or client) uses control information for integrity checking and secure communications. We present an algorithm for generating control information.

The remainder of this paper is organized as follows: Section 2 introduces preliminary notions that are needed throughout the paper. Section 3 describes the PSCS Protocol, and Section 4 presents the PSCS_{cp} protocol for a cache proxy. The complexity and security analysis is given in Section 5, and experimental results are presented in Section 6. We conclude the paper in Section 7.

2 Preliminaries

In this section, we introduce the notions and terminology used in our paper.

2.1 Content Service Functions and Privileges

Each content service belongs to a service function. The mapping from a content service to a service function is a many-to-one mapping. For example, a content service may compress images with less precision in order to reduce their size, or a content service may perform media conversion such as from text to audio or a format change such as from PDF to HTML. All these services belong to a transcoding function that changes the data from one format into another. We summarize the basic content service functions that intermediaries can perform in Fig. 2, which is an extension of [17]. We include some important classes of functions that are related to security services, such as the function of virus scanning.

To ensure data security, an intermediary must have certain privileges in order to access the data. Based on a client request, the data server decides the privileges for each participating proxy. For example, if a proxy needs to transcode the data from text to audio, then it needs to have certain privileges from the data server that authorizes this proxy to perform this transcoding function. Based on whether a service function needs to modify the requested data or not, we identify two types of privileges that allow intermediaries to perform content service functions: *read* and *update.* The *read* privilege allows a proxy to read and store the data. The *update* privilege allows a proxy to read and modify the data, as, for example, a proxy needs to have

```
<segment hash = encryptvalue1
delegateKey = pubkey1 delegateHash = encryptvalue2>
<segid> seg11 </segid>
This is the virus scan result.
</segment>
<segment hash = encryptvalue2>
<segid> seg18 </segid>
This is the data for virus scan.
</segment>
```

Fig. 3. Example of data segments.

this privilege in order to execute a content filtering function. It subsumes the read privilege. For each content service function, the corresponding privilege types are listed in Fig. 2.

2.2 Data Representation

We cast our approach in the framework of XML [9], [22] because of its widespread use in Web services. XML can be used to manage data, documents, graphics, and even multimedia data. Also, XML organizes data according to hierarchical nested structures, thus facilitating the parallelization. It organizes data into tagged elements. We define an *atomic element* (AE) as either an attribute or an element including its starting and ending tags. A data *segment* is a set of elements to which the same access control policy applies. That is, if a proxy has a read (or write) privilege over a segment, the proxy has a read (or write) privilege over all the elements in the segment. We enforce confidentiality by allowing a proxy to access only the segments that are permitted by access control policies. We assume that each segment is uniquely identified.

Based on the above concepts, we introduce our approach to data representation as follows:

Let $D = \{ae_1, ae_2, \ldots, ae_m\}$ be the data to be transferred, consisting of a set of AEs. Each AE is identified by an identifier. Data D are partitioned into a set of segments $\{Seg_1, Seg_2, \ldots, Seg_K\}$ such that

1. $\forall i \in \{1, \ldots, K\},\ Seg_i = (i, \{ae_{i_1}, ae_{i_2}, \ldots, ae_{i_r}\})$, each segment consists of a segment identifier (i) and of a set of AEs.
2. $\forall i \in \{1, \ldots, K\}$ and $\forall j \in \{1, \ldots, r\}, i_j \in \{1, 2, \ldots, m\}$, each AE in a segment belongs to D.
3. $\forall i \in \{1, \ldots, K\},\ \forall k, z \in \{1, \ldots, r\},$ if $k \neq z$, then $i_k \neq i_z$, AEs within the same segment are distinct.
4. $\forall i,\ k \in \{1, \ldots, K\}$ and $i \neq k,\ Seg_i \cap Seg_k = \emptyset$, AEs within disjoint segments are distinct.
5. For any $ae_i \in D, \exists j \in \{1, \ldots, K\}$ such that $ae_i \in Seg_j$, if an AE ae_i belongs to D, then ae_i must belong in a segment.

Properties 1, 2, and 4 ensure that there are a limited number of segments for the data. Property 3 ensures that the size of each segment is minimal. Property 5 ensures that the data is included in the segments. These properties ensure that the data is correctly represented by the set of segments.

To enforce authenticity and integrity, we rely on standard cryptographic primitives such as RSA public keys for digitally signing the data. Each segment has an encrypted hash value associated with it. If a proxy has an update privilege over a segment, when the proxy completes updating the segment, it generates a hash value by applying to the segment text, which also includes the segment identifier, a one-way hash function and then encrypts the value with its private key. Fig. 3 shows an example of data segments, which includes the result for virus scan and the data that is scanned. Attributes *delegateKey* and *delegateHash* are defined in Section 2.3.

P-proxy name	content service	Proxy/public key
P-proxy1	Virus scan	proxy1/pubk1, proxy2/pubk2
P-proxy2	Logo-adding	proxy3/pubk3
P-proxy3	Audio to text conversion	proxy4/pubk4, proxy5/pubk5
P-proxy4	Content filter	proxy1/pubk1

Fig. 4. Intermediary profile table.

2.3 Data Provider (DP) and P-Proxy

A DP is any entity that can provide the data requested by a client. Thus, a DP may be either a data server or a cache proxy caching the data requested by clients. In order to provide content services to clients, a DP has a group of cooperative intermediaries that can perform different content services.

A P-proxy is a list (size $\geq$ 1) of proxies that perform certain content services on the data on behalf of the DP. That is, for a DP, there may exist more than one cooperative proxy that can perform certain content services for it. Each DP maintains the information about the services provided by each cooperative proxy in an intermediary profile table. The intermediary profile table stores the public keys and the authorizations of proxies. Fig. 4 shows an example of such a table.

Because a proxy may provide several content services, it may appear in several different P-proxies maintained by a DP. In Fig. 4, proxy1 appears in both P-proxy1 and P-proxy4.

Even though a P-proxy may group several proxies, only one proxy in such group performs the content service associated with the P-proxy on each requested data. For example, suppose that proxy1 is a virus scan proxy in P-proxy1 and that P-proxy1 also includes proxy2. If proxy1 is overloaded, it can delegate to proxy2 the execution of the service. We refer to the proxy that is initially assigned to execute the operation on the data as the *primary proxy (prim)* of this P-proxy for the requested data. In the previous example, even though proxy2 executes the virus scan, the primary proxy is proxy1. The purpose of P-proxies is therefore to enhance both the availability and the efficiency of the system.

When a primary proxy p delegates the execution of the content services to another proxy q, where p and q belong to the same P-proxy, attributes $delegateKey$ and $delegateHash$ are required, where $delegateKey$ is q's public key, and $delegateHash$ is the digital signature of q signed with its private key on the digest of processed content. Note that q's public key is endorsed by p in p's signature.

2.4 Access Control System

Each DP has its own security policy related to its data. The access control system of each DP (Fig. 5) enforces which proxies and clients can access which data.

The inputs to the access control system include a client's request, the security policy and the intermediary profile table by the DP, and the data store. The access control system can return three possible access decisions:

1. *Deny.* This indicates that the DP does not have the data requested by the client, the client is not allowed to access the data according to the DP's policy, or no intermediaries in the DP's intermediary profile table

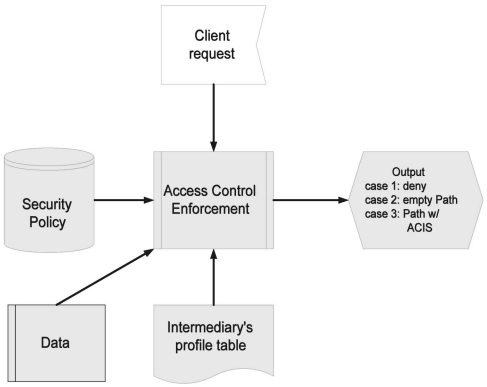

Fig. 5. Access control system.

exist or are allowed to transform the data into the version requested by the client.

2. *Empty path.* This indicates that the client's request can be satisfied without any intermediary's involvement.
3. *Path with ACIS.* This indicates that the client's request can be satisfied with the involvement of the P-proxies listed in the returned path. ACIS denotes *access control information structure*, which specifies the privileges over the data for each P-proxy in the path.

We now provide details concerning paths and ACIS.

A **path** denotes a *content service path* and explicitly defines the order according to which each P-proxy has to receive the data. That is, a path is a list of P-proxies. Let $P = \langle \text{P-proxy}_0, \text{P-proxy}_1, \text{P-proxy}_2, \ldots, \text{P-proxy}_{(N+1)} \rangle$ be a path such that

1. P-proxy_0 is the DP and $\text{P-proxy}_{(N+1)}$ is the client.
2. P-proxy_i $(i \in \{1, \ldots, N\})$ corresponds to a content service requested by the client.
3. If proxy $p \in \text{P-proxy}_i$ $(i \in \{1, \ldots, N\})$, then $p \in PT$, where PT is the P-proxy in the DP's intermediary profile table that performs the same content service as P-proxy_i. This requirement ensures that only proxies in the intermediary profile table are allowed to perform content services on the requested data.
4. If proxy $p \in PT$ and p is allowed by the DP's security policy to perform that content service on the data, then $p \in \text{P-proxy}_i$.

This requirement ensures that each P-proxy in Path includes all proxies that can perform that content service and also satisfy the security policy over the requested data.

Example 1. Suppose the following operations are to be performed on the requested data: virus scanning, logo adding, and audio-to-text conversion. The DP has an intermediary profile table as in Fig. 4, and its security policy allows these intermediaries to perform content services. The following content service path can be derived: $\langle \text{P-proxy}_0, \text{P-proxy}_3, \text{P-proxy}_2, \text{P-proxy}_1, \text{P-proxy}_4 \rangle$ which is illustrated in Fig. 6. As will be described in Section 3.1.2, a proxy (or client) is responsible for the integrity checking of the proceeding data transformation. Therefore, in Fig. 6, a cheating Proxy4 or Proxy5 will be detected and corrected by Proxy3. Note that for audio-to-text conversion, a malicious proxy may insert arbitrary text into the data. Because of the nature of the operation, it is very difficult for the next proxy (or the client) to determine whether the conversion is done honestly or some arbitrary text has been attached. The defense against such attack is out of the scope of this paper and remains an interesting open question.

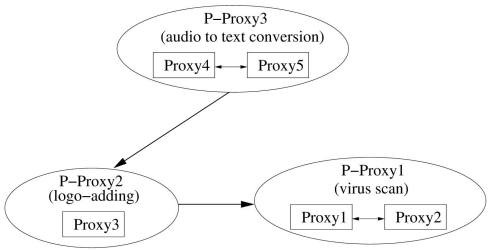

Fig. 6. Content service path example.

The requirements for the content service path are that the path should obey the same segment update-update order and update-read order. That is, if a segment is updated by content services i and j, the order of i and j is important. For example, in the previous example (Fig. 6), if segment seg is updated by both logo adding and audio-to-text conversion, then only after audio-to-text conversion can the logo be added to the segment. Thus, the content service dealing with text conversion must be placed before logo adding. Also, as the virus scan needs to read this segment, the virus scan must be placed after the logo-adding service.

Any content service path that satisfies the security policy and these order requirements can be used in our approach. The presence of more than one content service path for a request will not affect the control information (Section 2.5) generated for each P-proxy and the client.

Next, we explain the properties of ACIS. Let K be the total number of segments in the requested data. Let $\text{ACIS} = \langle ar_0, \ldots, ar_N, ar_{(N+1)} \rangle$ be the access control information structure such that

1. $ar_i = (readSet, updateSet)$, where $i \in [1, N]$; access segments for P-proxy_i in Path are split into read and update segment sets.
2. $readSet \subseteq \{1, \ldots, K\}$, $updateSet \subseteq \{1, \ldots, K\}$; the readSet (or updateSet) is a subset of the entire segments.
3. $readSet \cap updateSet = \emptyset$; if a segment is only readable for a P-proxy, then it cannot be in the updateSet of this P-proxy. If a segment is updatable for a P-proxy, then readability is implied, and there is no need to include the segment in the $readSet$.
4. $updateSet \cup readSet \subseteq \{1, \ldots, K\}$; the union of the sets is a subset of the entire segments.

For example, ar_0 is the access information for the DP. Thus, $ar_0.readSet = \emptyset$, and $ar_0.updateSet = \{1, \ldots, K\}$. $ar_{(N+1)}$ is the access information for the client. Thus, $ar_{(N+1)}.readSet = \{1, \ldots, K\}$, and $ar_{(N+1)}.updateSet = \emptyset$.

2.5 Control Information

The control information (CI, see Fig. 7) specifies which segments the primary proxy of a P-proxy in a path will receive and how the primary proxy can verify the integrity of each received segment. It also includes information on how to securely communicate with the P-proxy's successors ($succ$) and predecessor ($pred$). CI is a list. That is, $CI = \langle CI_0, \ldots, CI_N, CI_{(N+1)} \rangle$, where CI_0 is the control

$CI = \langle CI_0, CI_1, \ldots, CI_N, CI_{(N+1)} \rangle$ and
$CI_i = (i, pred, succ)$ is the control information generated for P-proxy$_i$ in Path
$pred = \{p_{P_1}, \ldots, p_{P_i}\}$: set of incoming package templates
$p_x = (pid,\ sk_{xi},\ seg)$: an incoming package template from P-proxy$_x$ in Path, where
1. $pid = x$ and $x \in \{P_1, \ldots, P_i\}$ pid is the sender's position in Path.
   $seg = \langle rs_1, \ldots, rs_{H(x)} \rangle$ where $rs_j = (r,\ s),\ j \in \{1, \ldots,\ H_{(x)}\}$ and $r \in \{1, \ldots, K\}$, s is the list of public keys of the last P-proxy p_y that modified r and the first element in s is the public key of the primary proxy in p_y.
   sk_{xi} is the symmetric key for encrypting/decrypting the package sending from P-$proxy_x$ to P-$proxy_i$.
2. $\forall j, w \in \{1, \ldots, H_{(x)}\}$: $j \neq w \Rightarrow rs_j.r \neq rs_w.r$ A segment must appear only once in the sequence of segments from a predecessor.
3. $\forall j, q \in \{1, \ldots, P_i\}$: $j \neq q \Rightarrow sk_{ji} \neq sk_{qi}$ $pred$ contains distinct predecessor P-proxies.
4. $\forall j, q \in \{1, \ldots, P_i\}$, $j \neq q$, $x \in \{1, \ldots, H_{(j)}\}$, $y \in \{1, \ldots, H_{(q)}\}$: $p_j.rs_x.r \neq p_q.rs_y.r$
   An accessible segment must be received only from one predecessor.

$succ = \{su_{S1}, \ldots, su_{Si}\}$: set of outgoing package templates
$su_y = (sid,\ sk_{iy},\ seg)$: an outgoing package template, where
1. $sid = y$ and $y \in \{S_1, \ldots, S_i\}$ *P-proxy(sid)* is the receiver of this package.
   sk_{iy} is the symmetric key as defined before.
   $seg = \langle r_1, \ldots, r_{W(y)} \rangle$: sequence of segments sent to *P-proxy(sid)*.
   $r_f \in \{1, \ldots, K\}$, $f \in \{1, \ldots, W_{(y)}\}$, $\forall j, g \in \{1, \ldots, W_{(y)}\}$: $j \neq g \Rightarrow r_j \neq r_g$ A segment must appear only once in the sequence of segments to be sent to a successor.
2. $\forall j, x \in \{S_1, \ldots, S_i\}$: $j \neq x \Rightarrow su_j.sk_{ij} \neq su_x.sk_{ix}$ Successor P-proxies are distinct.

Fig. 7. Control information specification.

information associated with the DP and $CI_{(N+1)}$ is the control information associated with the client. In Fig. 7, $H_{(x)}$ and $W_{(y)}$ are the numbers of components in the seg of p_x and su_y, respectively.

The control information CI_i for P-proxy$_i$ in a path also contains the corresponding incoming package templates and outgoing package templates. An incoming package template (p_x) contains the P-proxy's predecessor, which is P-proxy$_x$ in the path, and the set of segments (seg) that it will receive from this predecessor. The incoming package template also contains a symmetric key (sk_{xi}) for the receiver to decrypt the incoming package. For each segment R $(R \in seg)$ in the received package, the corresponding incoming package template includes the public keys $(R.s)$ of the last P-proxy that modified this segment (that is, the $(R.r)$th segment). A P-proxy uses the information contained in the incoming package template to verify that the received package is from a specified sender and that the data in the package is authentic up to that point. An outgoing package template (su_y) includes the successor, which is P-proxy$_y$ in the path, a symmetric key (sk_{iy}), and the set of segments (seg) to be sent to this successor. The receiver can then organize a package for the successor and encipher it with the symmetric key.

Each incoming package template has a predecessor ID (pid) associated with it. If a receiver receives from the same sender several packages at different times, the pid will help the receiver to determine which packages are referred to in its control information received from the DP. Each participant (intermediary or client) can use its control information for integrity checking and secure communications.

3 Parallel Secure Content Service Protocol

Before presenting the PSCS protocol, we give the security assumptions in our model:

- Each DP has a group of P-proxies that are *cooperative* with the DP in order to provide content services to clients.
- All proxies in a P-proxy are equal. That is, there is no proxy considered more trustworthy than another by the DP.
- All P-proxies that can perform the content services requested by a client are operative.
- They will execute their permitted update privileges correctly and will not collude with each other.
- If a P-proxy in a path cannot correctly perform the required content service, which involves updates, the request from the client cannot be satisfied.
- A P-proxy may attempt to access data without permission, thus violating data confidentiality, or it may attempt to modify data without permission, thus violating data integrity.

The PSCS protocol is a suite of protocols. It includes the PSCS Data Server (PSCS_{ds}) Protocol, which is responsible for handling requests by a data server, and the PSCS Cache Proxy (PSCS_{cp}) Protocol, which is responsible for handling requests by a cache proxy (CProxy) when it has a cache hit.

3.1 Parallel Secure Content Service Data Server Protocol

In the following, we discuss each participant's protocol and recovery protocol in detail.

3.1.1 Data Server Protocol

The data server sends the client's request to its access control system (a graph representation of the protocol is shown in Fig. 8). If the access decision on the request is deny, the data server notifies the client of the result. If the output is an empty path, the data server computes and signs the hash value of the data. Then, the data is transferred to the client using SSL/TLS [19], [21]. Both integrity and confidentiality are trivially satisfied in this case. We focus on a more complex case where the outputs from the access control system are a path and an ACIS.

Control information generation. The purpose of the control information is to help the P-proxies and the client to verify the integrity of the data and to securely communicate with each other. Based on the content service path and ACIS generated by the access control system, the data server uses the algorithm in Fig. 10 to generate the control information.

The algorithm is organized according to the following main phases:

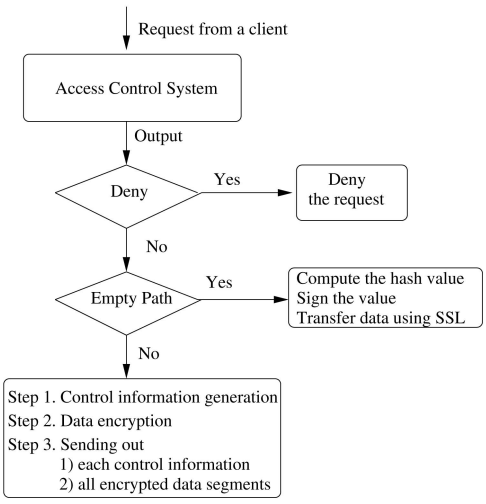

Fig. 8. Data server protocol for handling content service requests.

Initialization. First, for each participating P-proxy, the data server randomly orders the proxies in each P-proxy. The first one in the list is the primary proxy for this P-proxy. This random ordering avoids making certain proxies overloaded, especially when a proxy is being used in many P-proxies in a data server or a proxy appears in many DPs' intermediary profile table. This step also initializes each P-proxy's predecessors ($pred$), successors ($succ$) and segments (seg) that this P-proxy is authorized to access. See Fig. 7 for the notation used for $pred$ and $succ$.

Segment labeling. For each P-proxy in Path, this step labels each segment that this P-proxy is authorized to access with a list of public keys of the P-proxy that can modify this segment. The list of keys starts with the primary proxy of the P-proxy, followed by other proxies' public keys in the P-proxy. We use array $Seg[i].s$ to store the public keys of the P-proxy that most recently modified segment i and a structure ar_i that contains the set of accessible read and update segments of the $P\text{-}proxy_i$ in Path. Note that in step 2 in Fig. 10, $Seg[seg]$ is instantiated with different values for different $ACIS.ar_i$.

Generating the data server's CI. The data server needs to send segments to corresponding P-proxies or to the client. Each P-proxy must receive the segments that are allowed to access and send some or all of these segments to subsequent P-proxies or the client. For each segment, the data server scans the P-proxies according to the content service path; if a P-proxy needs to read this segment, then the data server adds this P-proxy to its $succ$, and this P-proxy will receive this segment from the data server. The data server repeats this activity until the first P-proxy that needs to update this segment. After the P-proxy updates the segment, it sends out the segment to the rest of the P-proxies or the client if they need to access it.

Function $add\text{-}pred(i, r, j)$ and $add\text{-}succ(j, r, i)$ are defined as follows: If there is no element $p \in P\text{-}proxy_j.pred$ such that $p.pid = i$, function $add\text{-}pred(i, r, j)$ inserts in the set $P\text{-}proxy_j.pred$ an element p, where 1) $p.pid = i$, 2) $p.sk = k$, where k is a symmetric key generated by the data server, and 3) $p.seg = \langle t \rangle$, where t is the tuple in $P\text{-}proxy_i.seg$ such that $\mathrm{t.r} = \mathrm{r}$. Otherwise, it appends t to $p.seg$. If there is no element $su \in P\text{-}proxy_i.succ$ such that $su.sid = j$, function $add\text{-}succ$ (j, r, i) inserts in the set $P\text{-}proxy_i.succ$ an element su, where

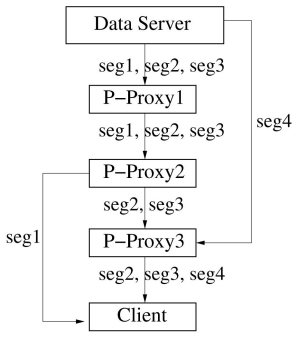

Fig. 9. An example of control information.

1) $su.sid = j$, 2) $su.sk = k$, where $k = P\text{-}proxy_j.pred.p.sk$, and 3) $su.seg = \langle r \rangle$. Otherwise, it appends r to $su.seg$.

Updating CI by calling procedure AddSeg. The procedure, reported in Fig. 11, updates CI by inserting segments that need to be sent out by each P-proxy (except the data server and the client) in the path.

Procedure $AddSeg$ (Fig. 11) works as follows: Let $P\text{-}proxy_i$ be a P-proxy that has updated a segment; it then sends it to the first subsequent $P\text{-}proxy_j$ in Path that can either read or update this segment (step 1). If the primary proxy of $P\text{-}proxy_i$ is different from that of the $P\text{-}proxy_j$ (step 1.a), the scan stops. If these two primary proxies are the same and $P\text{-}proxy_j$ has an update privilege over the segment again (step 1.b), then this segment will not appear neither in $P\text{-}proxy_i$'s $succ$ nor in $P\text{-}proxy_j$'s $pred$, and the scan stops. If the two proxies are the same and $P\text{-}proxy_j$ can only read the segment (step 1.c), $P\text{-}proxy_j$ will not receive the segment, and the scan will continue.

If $P\text{-}proxy_i$ has read access to a segment and needs to send it out (step 2), it will forward it to all subsequent P-proxies in the path that can only read the segment (step 2.b) until a P-proxy that can update the segment (step 2.a). Also, this last P-proxy will receive this segment. A P-proxy that receives a segment from another P-proxy such that both of them can only read the segment does not send the segment to any other proxy (step 2.b.b). If the segment is to be received later by $P\text{-}proxy_j$, whose primary proxy is the most recent one that updated it, we remove it from $P\text{-}proxy_j$'s $pred$ (step 2.b.a), and $P\text{-}proxy_i$ will not send the segment to $P\text{-}proxy_j$. If $P\text{-}proxy_j$ only has read access to the segment later and needs to send this segment to $P\text{-}proxy_x$, then the predecessor that is supposed to send the segment back to $P\text{-}proxy_j$ will send the segment to $P\text{-}proxy_x$. This does not only save bandwidth but also ensure data integrity. Finally, the control information CI is stored and returned. Note that the control information CI, not *ACIS*, is to be used for parallel processing among proxies.

Example 2. Suppose that P-proxy2 receives $seg1$, $seg2$, and $seg3$ from P-proxy1 and that these segments are updated by P-proxy1 (Fig. 9). The instructions from the data server to P-proxy2 are to transcode $seg1$ and send it to the client (no proxy will access $seg1$ anymore) and to form a new package that consists of two segments $seg2$ and $seg3$ and send it to P-proxy3. If P-proxy1 $= \langle proxy1/pubk_1, proxy2/pubk_2 \rangle$, P-proxy2 $= \langle proxy3/pubk_3,\ proxy4/pubk_4 \rangle$, and P-proxy3 $= \langle proxy5/pubk_5, proxy6/pubk_6,\ proxy7/pubk_7 \rangle$, where $pubk_i$ is the public key of $proxy(i)$, and $pubk_c$ is the public key of the client, then the control information for P-proxy2 is given as follows: $CI_2 = (2, pred, succ)$, where

Algorithm GenerateCI
Input: Path, ACIS
Output: CI

1. for each $i = 1$ to N
   P-proxy$_i$ = *randomly ordered proxies in P-proxy*$_i$
   P-proxy$_i.prim$ = *P-proxy*$_i$.head
   for each $i = 0$ to $N + 1$
   P-proxy$_i.pred = \emptyset$
   P-proxy$_i.succ = \emptyset$
   P-proxy$_i.seg = \emptyset$
2. for each $i = 1$ to K :
   $Seg[i].s = Path.pubk_0$
   for each $i = 1$ to $N + 1$
   $S = ACIS.ar_i$
   for each $seg \in S$
   add $(seg,\ Seg[seg].s)$ to *P-proxy*$_i.seg$
   if $seg \in S.updateSet$
   $Seg[seg].s$ = *P-proxy*$_i.pubkeys$
3. $\overline{ACIS} = ACIS$
   $AR = \overline{ACIS}.ar_0$
   for each $r \in AR$
   for $j = 1$ to $N + 1$
   if $r \in \overline{ACIS}.ar_j.updateSet$
   add-pred(0, r, j)
   add-succ(j, r, 0)
   break
   if $r \in \overline{ACIS}.ar_j.readSet$
   add-pred(0, r, j)
   add-succ(j, r, 0)
   $\overline{ACIS}.ar_j.readSet = \overline{ACIS}.ar_j.readSet \setminus \{r\}$
   continue;
4. AddSeg(Path, $\overline{ACIS}, CI$)

Fig. 10. Control information generation.

Procedure AddSeg
Input: Path, $\overline{ACIS}, CI$
Output: CI

for each $i = 1$ to N
$AR = \overline{ACIS}.ar_i$
1. for each $r \in AR.updateSet$
   for $j = i + 1$ to $N + 1$
1.a if $r \in (\overline{ACIS}.ar_j.updateSet \cup \overline{ACIS}.ar_j.readSet)$
   and Path.*P-proxy*$_j.prim \neq$ Path.*P-proxy*$_i.prim$
   add-pred(i, r, j)
   add-succ(j, r, i)
   break
1.b if $r \in \overline{ACIS}.ar_j.updateSet$
   and Path.*P-proxy*$_j.prim =$ Path.*P-proxy*$_i.prim$
   break
1.c if $r \in \overline{ACIS}.ar_j.readSet$
   and Path.*P-proxy*$_j.prim =$ Path.*P-proxy*$_i.prim$
   $\overline{ACIS}.ar_j.readSet = \overline{ACIS}.ar_j.readSet \setminus \{r\}$
   continue;
2. for each $r \in AR.readSet$
   for $j = i + 1$ to $N + 1$
2.a if $r \in \overline{ACIS}.ar_j.updateSet$
   add-pred(i, r, j)
   add-succ(j, r, i)
   break
2.b if $r \in \overline{ACIS}.ar_j.readSet$
2.b.a if Path.*P-proxy*$_j.prim \neq ar_i.Seg[r].s.prim$
   add-pred(i, r, j)
   add-succ(j, r, i)
2.b.b $\overline{ACIS}.ar_j.readSet = \overline{ACIS}.ar_j.readSet \setminus \{r\}$
   continue
for each $i = 1$ to N
CI$_i$ = (*i*, *P-proxy*$_i$*.pred*, *P-proxy*$_i$*.succ*)

Fig. 11. Procedure AddSeg.

- $pred = \{(1, sk_{12}, < (1, (pubk_1, pubk_2)),$ $(2, (pubk_1, pubk_2)),\ (3, (pubk_1, pubk_2)) >)\}$, and
- $succ = \{(4, sk_{24}, < 1 >), (3, sk_{23}, < 2, 3 >)\}$.

Once the data server has generated the control information and encrypted the data segments, it distributes them to the corresponding primary proxies and the client. The data server also provides a one-way hash function for the participating proxies and the client to verify the authentication of the data they received and to calculate a digest value for each segment they have updated.

Key management. We use public keys for signing and symmetric keys for encrypting contents. Although the number of keys used may be large, key management is quite simple, and there is no need for a public-key infrastructure and public-key certificates. The public keys of proxies are stored in the intermediary profile table maintained by the DP. This information is endorsed by the DP in its control information. The symmetric keys are preassigned by the DP and transmitted to corresponding proxies in encrypted form. Therefore, our protocol does not need any two-party key agreement protocol.

3.1.2 Intermediary Proxy and Client Protocol

During content service processing, the primary proxy in each P-proxy can either delegate the content services to another proxy in the same P-proxy or execute the function by itself.

If no delegation needs to be performed, the primary proxy executes a protocol that is similar to the client protocol. Such protocol has the following steps: check the integrity according to incoming package templates received from the data server, execute operations according to privileges, form package(s) according to outgoing package templates received from the data server, and send out packages. The details of these steps are given as follows:

Step 1. *Integrity check.* Upon receiving a package P, the receiver with control information CI_i verifies the following: 1) It verifies whether there has been any transmission error; if there is any such error, it asks the sender to send the package again. 2) It verifies that the package is from one of its predecessors. Suppose a receiver deciphers P with the symmetric key k such that $k = p_x.sk_{xi}$ and $p_x \in CI_i.pred$. If $P.pid \neq x$, the receiver discards the package. 3) It verifies the integrity and authorization of each segment according to the incoming package template. For each R in $p_x.seg$, the receiver checks if the segment in the package starts with a segment identifier equal to $R.r$. If so, the receiver generates a hash value using the one-way hash function, deciphers the hash in the package with $R.s$'s primary public key, and checks if these two values are equal. If the primary proxy of the P-proxy that authored the segment had delegated another proxy q for the update, then the receiver must check if q is in $R.s$ and $delegateHash$ is correct. If there is any error, the receiver asks the sender to recover.

Step 2. *Executing functions on the data.* After correctly receiving a package from each predecessor, the receiver executes its functions on the data. If it has update privileges on some segments, it will update the segments, calculate the hash value for each segment it updated, and cipher this value with its private key for future authorization checking.

Step 3. *Forming new package(s).* For each $su_y \in CI_i.succ$, the receiver forms an outgoing package U such that

Contents	Incoming package in CI_i	Outgoing package in CI_i	Total in CI
Symmetric keys	1	1	O(N)
Public keys	$O(1)$/segment	$O(1)$/segment	$O(NK)$
Segments	$O(K)$	$O(K)$	$O(NK)$

Fig. 12. The size complexity of control information.

$U.pid = i$. For each $r \in su_y.seg$, the receiver fills r in U. After this, the subject encrypts U with $su_y.sk_{iy}$ and sends it to the primary proxy of P-proxy_y. The receiver should also keep a copy for later recovery.

If a primary proxy decides to delegate its function to another proxy q in the P-proxy, the primary proxy first performs the integrity checking as in step 1. If there is no error, the primary proxy sends all the received packages to q, which completes the second step. q sends back only the updated segments with the $delegateHash$ attributes signed by q. The primary proxy calculates the hash value for each segment that is updated by q, makes sure that these values are equal to those signed by q, and then signs these hash values. At last, the primary proxy executes the last step of forming and sending out the outgoing packages.

3.1.3 Recovery Protocol

If a proxy receives a package that fails integrity verification, the proxy asks the sender to recover the package. If a receiver cannot get an error-free package according to the control information twice, it will send both packages it believes to be incorrect to the data server and the sender. The data server will delete this malicious proxy from its intermediary profile table. A simple solution is for the data server to broadcast to the P-proxies in the content services that the process has failed and aborted. However, this approach exposes the protocol to possible frequent failures. Indeed, if at least one proxy is malicious and generates a corrupted segment, the entire process fails.

To reduce such failures, the receiver will wait for a correct version from the data server. The data server will not broadcast an abort command immediately. It will first check if the malicious sender m of the error segment seg has only read access to this segment. If m only has read access, the data server will contact all the receivers that received seg from m. If any one has a correct version (passed integrity checking) from m, the data server will send this correct version to all the senders that did not receive a correct version from m. If no one has a correct version, the data server will ask the primary proxy of the P-proxy that authored this segment to send the data server a copy; the data server then acts the role of m, checking the integrity and sending this segment to all the receivers to which m was supposed to send it.

If m has an update privilege on the segment, the data server will check if there is any other proxy n in the P-proxy with m. If not, the data server aborts the process. Otherwise, it lets n execute the service m was suppose to execute. The receiver will then receive data from n.

4 Parallel Secure Content Service Cache Proxy Protocol

With cache proxies, a client's request is submitted to a cache proxy first, which may give a cache hit, that is, the requested content is cached by the proxy. In this case, the content is sent directly to the client without any processing. Cache hits can largely reduce the communication costs for delivering and computation costs for processing. If the requested content is not readily available at the cache proxy, the cache proxy handles the request as follows:

If a cache proxy is not the data server and can, however, satisfy a request, the cache proxy handles the request depending on whether it has a credential from the data server or not. If it has a credential that allows it to perform operations on requested data, the cache proxy handles the request and sends the requested data and the credential to the client. If the cache proxy does not have the proper credential, then it acts on behalf of the data server and does the following:

1. The cache proxy calls its access control system. If the request is denied, then the client is notified. Otherwise, the output of such a call includes a path P' and an ACIS. Note that P' should be part of the original path P if the client directly requests from the DP, that is, $P' \subseteq P$. An empty P' happens when the requested content is cached, which is then sent to the client. If P' is not empty, the cache proxy is the first one in the path that has update privileges on all the segments.
2. The cache proxy generates control information for the involved P-proxies and the client.
3. The cache proxy sends the control information to a data server.
4. If the data server allows the cache proxy to disseminate the data, the data server disseminates this control information to the corresponding intermediaries and the client; then, the data server asks the cache proxy to start sending out data segments. Otherwise, the data server replies to the cache proxy that the request is not allowed, and the data server handles the request of the client.

When a cache proxy handles the request, the amount of required bandwidth is greatly reduced, especially when a large amount of data is involved. Because adapted content is often buffered in proxies, cache consistency in cooperative proxies is an important problem. This problem is currently out of the scope of this paper, because we focus on the security (integrity and confidentiality in particular) of data adaptation and intermediary caching. We refer readers to existing literature on general cache consistency problem for more information.

5 Complexity and Security Analysis

For complexity analysis, we focus on the size and generation time of control information, the complexities of which are shown in Figs. 12 and 13, respectively. Let N be the total number of proxies required to process a request, and $i \in [1, N]$. Let K be the total number of segments in the request content. The analysis of time complexity follows the description of control information generation in Section 3.1.1.

Initialization	Segment Labeling	Generating *CI*	AddSeg	Total running time
$O(N)$	$O(NK)$	$O(NK)$	$O(N^2K)$	$O(NK + N^2K)$

Fig. 13. The time complexity of preparing control information.

The analysis assumes that there are a constant number of proxies in each P-proxy. In the next section, we give the overall performance results of our protocols.

We analyze the security for the case that a client's request is handled by a data server. Due to space limitations, the analysis about the security of the PSCS_{cp} Protocol is not provided, as it is similar to the analysis of the PSCS_{ds} Protocol. Our protocol assumes a digital signature scheme that is secure against existential forgery attacks. Intuitively, this means that a polynomial-time adversary cannot successfully forge the signature of a message that a signer has not signed. We also assume a symmetric key encryption scheme with one-way security, which intuitively means that a polynomial-time adversary cannot successfully guess the plaintext after seeing the ciphertext.

Theorem 1. *The* PSCS_{ds} *Protocol is secure with respect to integrity.*

Proof. We need to prove that a proxy cannot update a segment over which it does not have update privileges. If a primary proxy delegates another proxy q in the P-proxy for the execution of content services, according to the protocol executed by intermediary proxies, proxy q cannot modify a segment that it is not allowed to update and send the segment out.

Thus, we only need to consider the primary proxies. There are two cases:

Case 1. *A primary proxy modifies a segment that is not authored by itself, and the proxy does not have the authorization to update the segment.*

In this case, integrity is enforced by digital signatures. When a proxy i updates a segment seg, it signs the hash value that it generates from seg with its private key. If a primary proxy j has *read* authorization on seg, j receives control information from the data server, which contains an incoming template. The incoming template includes the public key of i for deciphering the hash. j will calculate the hash of the segment and check the signature. Suppose an adversarial proxy can modify segment seg before it reaches j and the signature on the modification is accepted by j. This means that the adversarial proxy can successfully forge i's signature and thus break the digital signature scheme. This contradicts with the assumption on the security of the signature scheme.

If a primary proxy m receives two segments authored by the same proxy i, it cannot switch the information in these two segments, as a segment represented in XML has the segment identifier in its tag and this identifier is included when the hash value is generated.

Although a primary proxy may not indicate that it delegates another proxy for the update of a segment, this does not violate the integrity, because the primary proxy also has the privilege to update the segment. It can write the segment with the result of the delegated proxy.

Thus, no proxy can modify a segment that has not been authored by itself and for which it does not have an update privilege.

Case 2. *A primary proxy modifies a segment authored by itself, even though it does not have update authorization over it now.*

First, we discuss the case in which no delegation happens. Suppose a segment seg is updated by proxy A, then read by proxy B, and, after that, sent by B to A for reading (A cannot update seg this time) and then by B to C for reading. In this case, A cannot send C a segment that is different from the one it sends to B. As in the control information generation, B will send a copy to C instead of A. Thus, C receives the segment that A authored at the beginning.

In the case of delegation, suppose a segment seg is updated by A, then read by B, and then distributed by B to C for reading. Suppose A delegates to B, updating seg; after B finishes updating, A also needs to sign the hash value of the segment. B cannot change seg when accessing it the second time, because C will also verify the segment with A's signature as A is the primary proxy.

Thus, data integrity is enforced. □

Theorem 2. *The* PSCS_{ds} *Protocol is secure with respect to confidentiality.*

Proof. We need to prove that if a proxy is not authorized to access a segment, then it cannot access it. During the communication, the confidentiality is enforced by the symmetric key that enciphers/deciphers a package so that only the designated receiver can access it. When a proxy receives a package, it uses the control information received from the data server to decipher the package. Suppose an adversarial proxy has successfully deciphered the package and broken the confidentiality. This means that the adversary has broken the symmetric key encryption scheme, which contradicts with the assumption of a secure symmetric key encryption scheme. Besides, the control information generation ensures that a proxy only receives the segments that it is authorized to access. Thus, the protocol is secure with respect to confidentiality. □

It is important to notice that under our approach a receiver could learn the access privileges of its predecessor(s) or successor(s). The disclosure of such access privileges, however, does not violate confidentiality and integrity, because these requirements only concern the data contents.

6 Experiments and Results

In this section, we report some results for the experiments we have carried out to evaluate the performance of our approach. The primary goals of our experiments are to understand the characteristics of the PSCS Protocol with respect to three properties:

1. *Size of transmitted data.* how much bandwidth our approach saves?

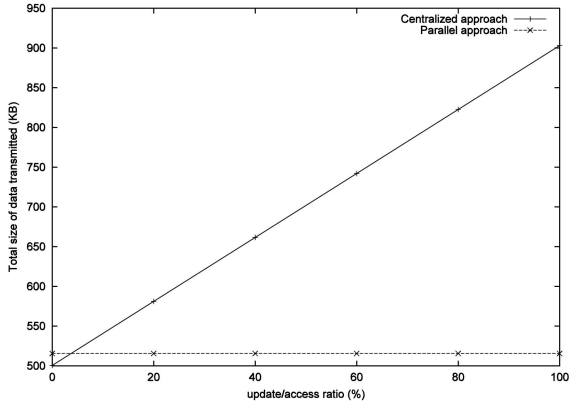

Fig. 14. Comparison of the total size of data transmitted.

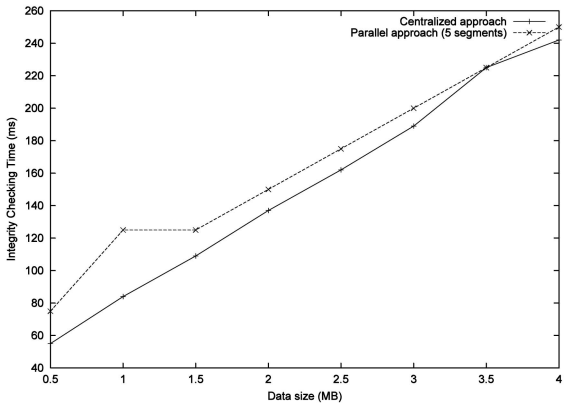

Fig. 15. Comparison of the integrity check time.

2. *Integrity checking time.* how much does this security requirement cost?
3. *Content servicing time.* how much could the PSCS Protocol save in the overall servicing time and how does the recovery affect the overall servicing time?

We compare our approach with a centralized approach. In the centralized approach, the data server sequentially contacts each P-proxy to complete the requested content services. Once an intermediary has completed the operation, it sends back only the data that it has updated to the data server, and the data server then prepares the data for the next P-proxy. Both the server and intermediaries use message digests and digital signatures to ensure data integrity, and they communicate through a secure channel. Once the data server receives the data from the last intermediary, it finally sends the data to the client. The centralized model has a star-shaped communication pattern, where the data server is at the center, and each proxy only communicates with the data server. In addition, the communication is always a round-trip traffic, that is, from the data server to a proxy and then back.

The experiments have been carried out in a high-speed local area network with an average bandwidth of about 100 Mbps. The data server, client, and intermediaries are on the same network, and each machine runs Linux OS with a Pentium III 500-MHz CPU.

6.1 Data Size

Reducing the total size of the data transmitted in the system could improve system performance by reducing congestion and collision. In a low-bandwidth system, the time difference between transferring a large amount of data and a small amount of data is quite significant. First, we should notice that even if the content is divided into different segments such that the sum of all segment sizes is equal to the size of the whole content, for a symmetric encryption algorithm, the size of the ciphertext is the same as the cleartext. Thus, the ciphertexts under these two approaches have the same size. Second, in XML encryption, the cleartext uses UTF-8 encoding (8 bits per character), whereas the ciphertext is base64 (6 bits per character) text. Thus, the ciphertext is about 1.33 times of the size of the cleartext. Enforcing the confidentiality of XML data during communications requires transmitting more data.

Fig. 14 reports the total size of the data transmitted *by all entities* in the system under the two approaches when no recovery is executed. In the experiment, we considered the case of 100-Kbyte data equally divided into 25 segments. All segments are accessed by four P-proxies, and RSA signatures of size 128 bytes are used. From the graph, we can see that our approach requires transferring less data than the centralized approach when the *update/access* ratio is higher than 4 percent. Because most content service functions (see Fig. 2) require an update privilege that has an *update/access* ratio of 100 percent, our approach is much more efficient than the centralized approach with respect to the amount of data to be transmitted on the network.

In fact, our approach requires transferring a minimal amount of data with very low overhead. Such overhead basically arises because of the signatures associated with the segments. For n P-proxies, there are at most 3^n segments, because for each segment, a P-proxy can have either read or update access or no access to the segment. For large amounts of data and a few segments in the data, the additional data due to the signatures is almost negligible.

6.2 Integrity Check Time

Since our approach requires each P-proxy to perform an integrity check, we have evaluated the time of this check by dividing a whole document equally into five segments. For the parallel approach, the time of the check is given by the sum of the times required for computing the hash values of the five segments and for the RSA signature verification. Under the centralized approach, only one hash value is computed for the whole document, and only one RSA signature verification has to be executed. Fig. 15 reports the time for different data sizes.

As the time to compute a hash value is almost linear in the size of the data and because verifying a digital signature is quite fast, our results indicate that our approach does not have much additional overhead with respect to the centralized approach (less than 50 ms for a data size of 4 Mbytes).

6.3 Overall Content Servicing Time

First, we investigate how much our approach could save in the overall content servicing time against the centralized approach. The experiments have been run with four proxies (as data normally require about four different content services), and all segments have the same size ($\text{size} = \text{N}/\text{M}$, where N is the size of a document, and M is the number of segments). Each proxy transcodes the segments it receives from the data server, and then, these results are sent to the client. We use a ratio to represent the data size distribution.

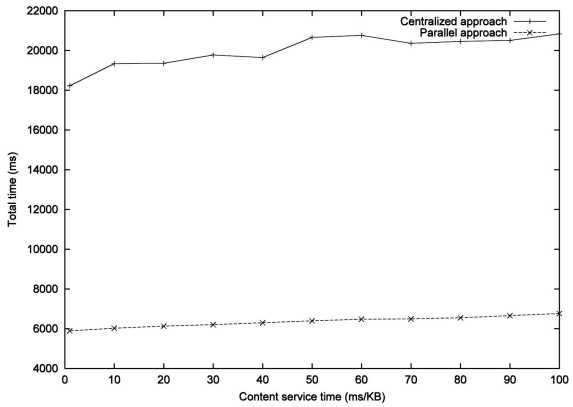

Fig. 16. Comparison of the overall servicing time (data size = 11 Kbytes).

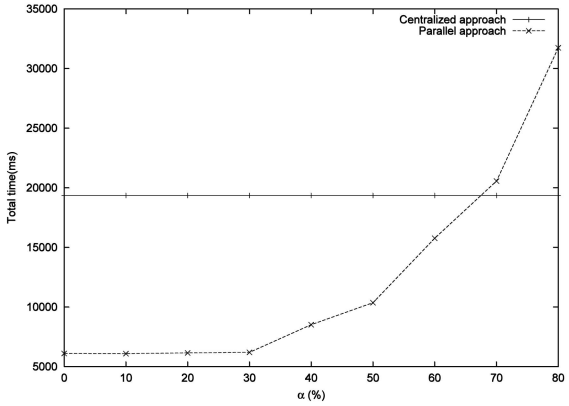

Fig. 18. Comparison of the overall servicing time in case of recovery (data size = 11 Kbytes).

For simplicity, in our experiment, we assume that the segment distribution among the client and four proxies is 1:1:2:3:4. Fig. 16 reports the overall servicing times for data of size 11 Kbytes.

In the figure, we can see that if the operation time for a content service is 1 ms/Kbyte, our approach can save at least 12 sec. As the size of the data increases (see Fig. 17), the savings obtained by our approach increase. Thus, even though our approach spends a little more time during integrity checking, overall, it is much more efficient than the centralized approach. Figs. 16 and 17 also show that as the operation time increases, the overall servicing time of our approach is much lower than that of the centralized approach. As noted by previous research [6], [12], transcoding normally involves intensive computation. In multimedia applications, our approach is thus highly desirable.

Next, we investigate the effect of recovery. Fig. 18 reports the comparison of the overall servicing time in the case of recovery. As we can see in the figure, the operation time for a content service is 20 ms/Kbyte. The *access/update* ratio for each proxy is 10 percent, and each proxy also needs to send its access segments to the client. We assume that the probability α that a proxy performs illegal modification has a uniform distribution. As α increases, the PSCS protocol spends more time in recovery, and the overall servicing time may be higher than the servicing time of the centralized approach.

Next, we investigate the effect of recovery under different data sizes. Fig. 19 reports the comparison of the overall servicing time in the case of recovery for data of size 22 Kbytes. All the other parameters are the same as that of the experiments reported in Fig. 18. By comparing Fig. 19 with Fig. 18, we can see that if the data size is larger, the α value required to make the PSCS Protocol take a longer overall servicing time than the centralized approach is lower. This is because the data transmitting time dominates the overall servicing time.

Our protocol still has performance gains even in the case where no parallel service exists, because proxies have the proper control information to sequentially process the content among themselves without reporting to the DP. This saves communication costs compared to a centralized approach, where the content is returned to the DP after each transformation.

7 Conclusions

In this paper, we have presented a solution for secure content services characterized by a scalable and robust network architecture. Our protocol allows a client to verify that the received data is authentic and transformations on the data are properly authorized. Our approach also assures data confidentiality during transmission. It highlights load distribution through P-proxies to improve system performance and supports parallel content services. Because no modification is required to current content distribution

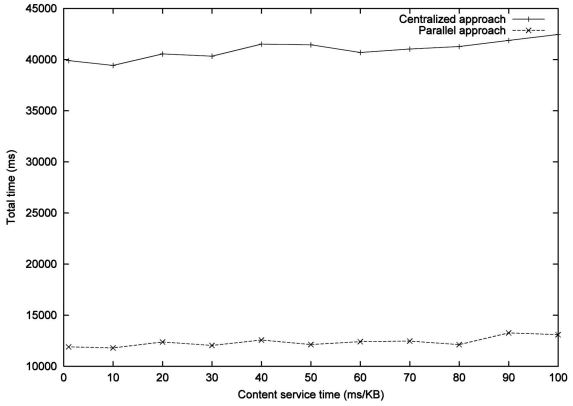

Fig. 17. Comparison of the overall servicing time (data size = 22 Kbytes).

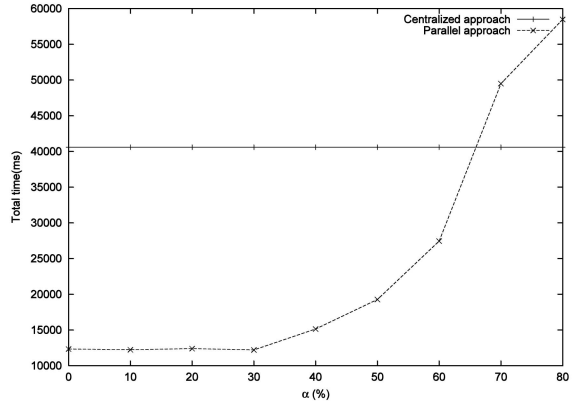

Fig. 19. Comparison of the overall servicing time in case of recovery (data size = 22 Kbytes).

systems in order to adopt our approach, our work is easy to deploy for many applications. In addition, our approach is extensible; if a new type of content service is required, our architecture can be easily adapted to the new requirement.

REFERENCES

[1] C. Aggarwal, J.L. Wolf, and P.S. Yu, "Caching on the World Wide Web," *IEEE Trans. Knowledge and Data Eng.*, vol. 11, no. 1, pp. 94-107, Jan. 1999.

[2] G. Berhe, L. Brunie, and J.M. Pierson, "Modeling Service-Based Multimedia Content Adaptation in Pervasive Computing," *Proc. First Conf. Computing Frontiers,* Apr. 2004.

[3] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM '99,* Mar. 1999.

[4] S. Buchholz and A. Schill, "Adaptation-Aware Web Caching: Caching in the Future Pervasive Web," *Proc. 13th GI/ITG Conf. Kommunikation in Verteilten Systemen (KiVS),* 2003.

[5] V. Cardellini, P.S. Yu, and Y.W. Huang, "Collaborative Proxy System for Distributed Web Content Transcoding," *Proc. Ninth ACM Int'l Conf. Information and Knowledge Management (CIKM '00),* Nov. 2000.

[6] S. Chandra and C.S. Ellis, "JPEG Compression Metric as a Quality-Aware Image Transcoding," *Proc. Second Usenix Symp. Internet Technology and Systems (USITS '99),* Oct. 1999.

[7] C.H. Chi, Y. Lin, J. Deng, X. Li, and T. Chua, "Automatic Proxy-Based Watermarking for WWW," *Computer Comm.,* vol. 24, no. 2, pp. 144-154, Feb. 2001.

[8] C.H. Chi and Y. Wu, "An XML-Based Data Integrity Service Model for Web Intermediaries," *Proc. Seventh Int'l Workshop Web Content Caching and Distribution (WCW '02),* Aug. 2002.

[9] *Extensible Markup Language (XML),* http://www.w3.org/XML/, 2007.

[10] A. Fox, S.D. Gribble, Y. Chawathe, and E.A. Brewer, "Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives," *IEEE Personal Comm.,* Aug. 1998.

[11] M.J. Freedman, E. Freudenthal, and D. Mazières, "Democratizing Content Publication with Coral," *Proc. Usenix/ACM Symp. Networked Systems Design and Implementation (NSDI '04),* Mar. 2004.

[12] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas, "Dynamic Adaptation in an Image Transcoding Proxy for Mobile Web Browsing," *IEEE Personal Comm.,* vol. 5, no. 6, pp. 8-17, Dec. 1998.

[13] J-L. Huang, M-S. Chen, and H-P. Hung, "A QoS-Aware Transcoding Proxy Using On-Demand Data Broadcasting," *Proc. IEEE INFOCOM '04,* Mar. 2004.

[14] Y. Koglin, G. Mella, E. Bertino, and E. Ferrari, "An Update Protocol for XML Documents in Distributed and Cooperative Systems," *Proc. 25th Int'l Conf. Distributed Computing Systems (ICDCS '05),* June 2005.

[15] B. Li, M.J. Golin, G.F. Italiano, X. Deng, and K. Sohraby, "On the Optimal Placement of Web Proxies in the Internet," *Proc. INFOCOM '99,* Mar. 1999.

[16] W.Y. Lum and F.C.M. Lau, "On Balancing between Transcoding Overhead and Spatial Consumption in Content Adaptation," *Proc. ACM MobiCom '02,* pp. 239-250, Sept. 2002.

[17] P. Maglio and R. Barrett, "Intermediaries Personalize Information Streams," *Comm. ACM,* vol. 43, no. 8, pp. 99-101, Aug. 2000.

[18] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M.V. Steen, "Replication for Web Hosting Systems," *ACM Computing Surveys,* vol. 36, no. 3, pp. 291-334, Sept. 2004.

[19] *SSL Specification,* http://wp.netscape.com/eng/security/SSL_2.html, 2007.

[20] B. Thuraisingham, A. Gupta, E. Bertino, and E. Ferrari, "Collaborative Commerce and Knowledge Management," *Knowledge and Process Management,* vol. 9, no. 1, pp. 43-53, Aug. 2002.

[21] *Transport Layer Security,* http://www.ietf.org/html.charters/tls-charter.html, 2007.

[22] *W3C XML Schema,* http://www.w3.org/XML/Schema, 2007.

Yunhua Koglin received the PhD degree in computer science from Purdue University in 2006. She is now with Cisco System. Her research interests are secure content distribution, access control, and applied cryptography.

Danfeng Yao is a PhD candidate in the Computer Science Department, Brown University. She has interned in the Trusted Systems Laboratory, Hewlett-Packard Laboratories, and worked in the Center for Education and Research in Information Assurance and Security (CERIAS), Purdue University. She will join the Department of Computer Science, Rutgers University, New Brunswick, as an assistant professor in January 2008. Her research interests are in information security and applied cryptography. She is a member of the IEEE. She served as a PC member in the Eighth IEEE SMC Information Assurance Workshop in 2007. She won the Best Student Paper Award in ICICS 2006 and the Award for Technological Innovation from Brown University in 2006. She has two US patents pending for her work on identity management.

Elisa Bertino is a professor of computer science at Purdue University and serves as the research director of the Center for Education and Research in Information Assurance and Security (CERIAS). Previously, she was a faculty member in the Department of Computer Science and Communication, University of Milan, where she has been the department chair and the director of the DB&SEC Laboratory. Her main research interests include security, privacy, database systems, object-oriented technology, and multimedia systems. She has published more than 250 papers in major refereed journals and in the proceedings of international conferences and symposia. She has coauthored three books. She is a co-editor in chief of the *Very Large Database Systems Journal.* She serves also on the editorial boards of several scientific journals, including *IEEE Internet Computing*, the *IEEE Transactions on Dependable and Secure Computing*, *IEEE Security & Privacy*, and the *ACM Transactions on Information and System Security.* She is a fellow of the IEEE and the ACM and has been named a Golden Core Member for her service to the IEEE Computer Society. She received the 2002 IEEE Computer Society Technical Achievement Award for "outstanding contributions to database systems and database security and advanced data management systems" and the 2005 IEEE Computer Society Tsutomu Kanai Award for "pioneering and innovative research contributions to secure distributed systems."