

An XML-Based Data Integrity Service Model for Web Intermediaries

Chi-Hung Chi, Yin Wu
School of Computing
National University of Singapore
Email: chich@comp.nus.edu.sg

Abstract

Recent research in active network and web intermediary services result in web contents being disassembled, transformed and recomposed in a pipelined process as they flow through the Internet. This makes the web pages more vulnerable than they were ever before. Thus, there is an urgent need for a formal model to define and protect data integrity of web contents. In this paper, we propose a **Data Integrity Service Model (DISM)**. DISM is an XML language designed to be interposed into markup to preserve data integrity of the web contents. It allows the content owner to express their intentions, in terms of permissions, as part of the response message; it allows the proxies to execute and delegate the owner's permissions; and it allows any party to validate the page content with respect to the owner's permissions. The W3C XML signature standard serves as a foundation for the authorization and validation system in DISM. Furthermore, the model is friendly to proxy caches in which fine-grained partial contents can be cached. It provides a complete set of mechanisms to request and invalidate the cached contents.

1. Introduction

With the proliferation of the intermediary web services such as cache, request redirection, content filtering, protocols trying to unify the semantics of these services emerge (e.g. ICAP [2], SOAP [3] and WSDL [4]). The reasons we need intermediary services are three fold:

1. The bottleneck of the Internet is usually in the middle of the traffic. This is typical when we have robust server and clients but poor Internet connection. In this case, caching is a solution far better than increasing the processing capability of the server and the clients.
2. It is more cost-effective to put some services in the middle. Services like webpage personalization, local advertisement insertion can be carried out at any point in the Internet. However, it maybe less expensive to do this on the gateway near the client simply because 1) It is very difficult for the server to maintain a vast database for all the user profiles, 2) Low-end clients may not be able to process these services (e.g. PDA, handphone).
3. Some service cannot be carried out either on the server or on the client. One typical example is content filtering. Filters on the local computers are proved to be easily broken. As for the filter on the sever side ... How can you expect the Playboy magazine to filter its own content to echo the parents accusation?

For all these reasons, the proxy becomes the centre of business interest because of its special position and its

potential processing power. Today's proxies do much more than simple caching. They enforce access policy, they authenticate users, they present a uniform view of federated websites, they route requests through content delivery networks, they customize web pages, and they offload backend processing.

While providing these services, proxies make change to the delivered content. However, not all proxies are owned and administered by the content owner. Hence, the right to modify the content is implicitly delegated to other parties without proper limitation. The prevailing priority of the proxies over the clients make this "unlimited" modification right serious enough to undermine the data integrity of the web pages – the clients have no way to validate the data they received!

The most straightforward solution to this problem is to encrypt every bit in the communication so that to keep it secret from the intermediary proxies (E.g. The SSL [6]). However, this solution preserves data integrity at the price of losing all the valuable services. Since the communication is secret to the proxies, there is no way for them to cache and reuse the server responses. Other value-add services are also impossible. Therefore, what we need is a formal model addressing both the data integrity and the active-network services. The following are the features that an ideal model should possess:

- The model should be secured so that it is computationally infeasible to tamper the web content.
- The model should be open to value-added services.

- The model should be compatible to the popular protocols (e.g. HTTP1.1 [5])
- The model should be lightweight and it is preferred that it supports cache.

1.1. Related Works

Among the existing protocols concerning Web services, Security Socket Layer (SSL) is the only one that provides protection to data integrity. SSL relies on the concept of a secured channel. This channel guarantees confidentiality in that all messages that pass over it are encrypted. The SSL technology provides two functions: encrypting the information flow between client and server and forming the basis for mutual client/server authentication. However, the nature of absolute confidentiality of SSL makes it unsuitable for the server-proxy-client infrastructure. SSL secures the data stream from the TCP/IP layer, eliminating the interference of any proxy server. Therefore, proxy caching for SSL is unavailable. Other value-added service like webpage customisation, virus scan etc. are also impossible. Another drawback of SSL is that it does not support pre-processing. Data are encrypted when the request arrives and a new encryption session must be created for each incoming request. This seriously limits the server response speed especially when the number of requests is vast. There are also other calls for data integrity model [7]. However, no formal definition is given so far.

2. Overview of DISM

In this paper, we propose an XML-based solution to address data integrity. DISM is an in-markup XML-based language designed to preserve the integrity of the web pages. DISM is compatible with HTTP protocol. It accepts normal HTTP requests; and the DISM response message can pass through non-DISM proxy without causing any error. The functions of DISM can be summarized as the following:

- DISM allows the content owner to specify modification policies on the web contents. These policies indicate when, how and by whom the content can be changed. The policies can be attached to the server's response message and delivered together with the content. DISM also allows these policies to be delegated to other trusted parties.
- DISM provides an authorization and validation model to protect the response message from being tampered. This security model is based on the pub-

lic-key cryptosystem. It allows any party to validate the message with respect to the server's digital signature. The W3C XML signature standard serves as a foundation for the definition of this model.

- DISM allows the content owner to divide the web content into separate parts and assign each part different cacheability profile. It also allows proxies to treat parts of web objects as cacheable resources. This gives the client the flexibility to request for partial cache invalidation, therefore, reducing both the required bandwidth and the server workload.

In DISM, the basic unit for authorization, modification and validation is "segment". A web page is divided into a set of segments each of which possesses its own content, modification rights and modification record. The collection of these segments is a "manifest".

The modification rights of the segments are called "permissions". Permissions specify how their parent segment can be modified. A complete permission contains the information about type of the permitted "action" (E.g. add, delete, replace), conditions under which the permission can be executed, identity of the authorized "editor" and the authorizer's digital signature. Some permission can be further delegated to other trusted party if the "delegatable" attribute is set to be "yes". Nevertheless, issuing new permission is beyond the right of the proxies. The security policy of DISM is "monotonic" in the sense that the number available permissions can only decrease as the message moves between the editors. The content owner, normally the server, is the only one that has the privilege to give new permissions.

In order for the client to validate the received message, each editor must leave a modification record in the corresponding segment. These modification records indicate when, how and by whom the content was modified. The modification records are secured by the editor's signatures. DISM is a cache-friendly model in the sense that the segments are relatively independent of each other and they can be separately cached and invalidated.

2.1. W3C Digital Signature Standard

In this paper, we use the term "signature" or "digital signature" to refer to authentication values defined by the W3C XML-Signature Syntax and Processing standard, which are based on the public-key cryptosystem and which provide signer authentication. The XML Signature is a method of associating a key

with referenced data (octets). XML Signatures can be applied to any digital content (data object), including XML elements.

The digital signature serves as the base of the authorization and validation model in DISM. The definition of DISM's "Signature" element is directly borrowed from the W3C standard. It secures the web content as well as the server's modification permissions. It allows any party to validate the received message with respect to the server's public key. In this paper, since we focus on the functions of DISM, we will not explain the theory of signatures into details. Please refer to the [1] for the specifications.

2.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Publisher: The one who provides the content. In most of the cases, the terms "publisher", "content owner" and "server" are equivalent.

Delegate: The one who is authorized by the publisher to modify the content or delegate the permissions.

Editor: The one who modifies the content.

DISM message: An XML-based server response conforming DISM standard. A DISM message contains not only the page content but also the permissions and modification records.

DISM processor: Any logic unit that can understand, execute and validate DISM messages.

3. The DISM Data Flow

DISM is a simple markup language defined to preserve data integrity of web objects. Each editor must leave a modification record and his digital signature in the modified part. The receiver can validate the content using these records and their corresponding signatures. DISM does not prevent the content from being tampered. However, the unauthorized editor cannot leave a valid signature since he is not able to present the server's authorization. The server expresses its authorizations in terms of permissions. Permissions specify the identity of the authorized party as well as his permitted action.

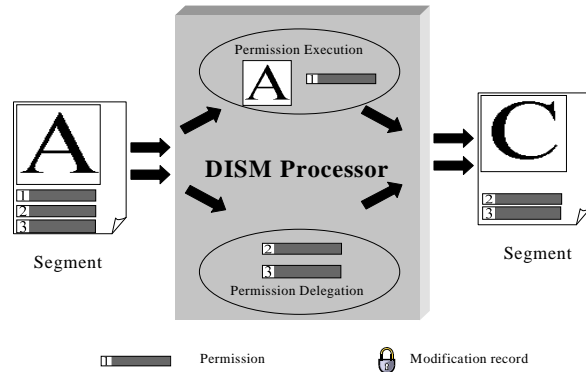


Figure 1: Processing of Segment

In a DISM message, the page content is divided into a set of segments each of which is associated with some permissions. (See Figure 1) When a segment flows through the DISM processor, the processor picks up its assigned permission and performs the corresponding modification. However, the processor is not obliged to execute the permission. A processor has the freedom to ignore its designated permission if it is either unable or unwilling to execute the permission. A processor also has the option to delegate the permission to other parties if the permission is "delegatable". It is important to note that each permission can only be executed once and no further delegation is allowed for executed permissions. The security policy in DISM is "monotonic", meaning that the number of available permissions can only decrease as the message moves between editors.

3.1. DISM Message Validation

To allow the value-added services, DISM does not explicitly encrypt the content of the web page. Instead, DISM requires each editor to leave a modification record in the message so that the receiver can validate the content with respect to these records. Although message validation is optional, the editors are recommended to validate the content before performing the modification so as to ensure that their modifications are based on valid content.

The modification records are secured by the editor's signature. The record contains the details of current modification, the authorizing permission, the digital signature and (conditionally) the modification record left by previous editors. The authorizing permission shows the legitimacy of the modification. The modification details indicate how the content was changed. The previous modification record testifies that the modifica-

tion is based on valid content. Since all the information needed for the validation is enclosed in the modification record (assuming that the public keys of the server and editors are known to everybody), the content validation can be carried out locally, therefore, saving the round-trip time.

Each new record refers to its preceding record, if any, forming a record chain. If any node of this chain is invalid the validation will fail. The first node of this chain is always the server's modification – publishing the content. If the validation fails, the receiving party have two options: 1) discard the invalid message and make a reply of failure (either HTTP 404 "page not available" or HTTP 500 "server error"); 2) mark the invalid part and forward it. The editor should not do further modification after finding the content to be invalid.

3.2. Cache In DISM

DISM allows fine-grained cache of content segments. This enables the client to invalidate specific parts of the content, reducing bandwidth consumption as well as the server/proxy workload (Less segments need to be processed). It also gives the server/proxy an option to pre-process and cache some of the frequently requested segments so that to balance the workload between the peek hours and off-peek hours.

In DISM, content segments are relatively independent of each other. Each segment possesses its own permission set, modification record and the cacheability profile. This gives the proxy great flexibility to cache the content segments as separate elements since removing, modifying or invalidating one of the segments does not affect the status of the others.

DISM defines a complete set of mechanisms for cache invalidation. Unlike the normal "if-modified" HTTP request, the DISM invalidation request is submitted by using the HTTP "POST" command. The procedure is very similar to blank filling. (See Figure 2) The client posts an invalidation form listing the IDs of the expired segments. As the form flows all the way to the server, each proxy tries to invalidate the listed segments and passes the remaining to the next proxy. When all the segments are invalidated, the form will be returned to client immediately.

DISM provides an open invalidation system in the sense that the expiration factor is not limited to the traditional

one – time. It can be any factor that could possibly make the content stale (E.g. user group). The syntax expressing these factors is open to change. Everybody has the freedom to introduce new factors as long as the designated server can understand it. However, factors that are not understood by the server/proxy are ignored, and the default result of the invalidation with unknown factor is "content-changed".

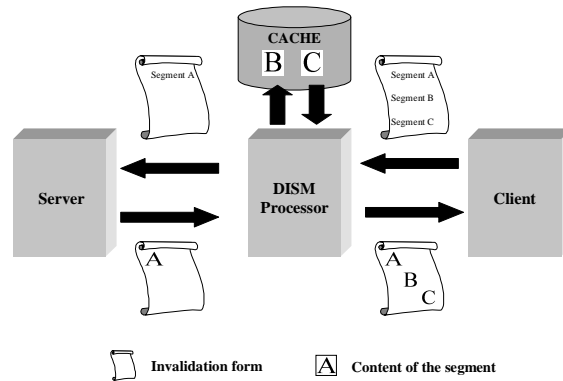


Figure 2: Cache Invalidation

4. DISM Elements

In a DISM message, the content of a web page is divided into a set of segments. The collection of these segments is the manifest. Each segment is associated with some modification rights – permissions. Permissions specify how the segments can be modified. A complete permission contains the information about type of the permitted action (E.g. add, delete, replace), conditions under which the permission can be executed, identity of the authorized "delegate" and the authorizer's digital signature. Some permission can be further delegate to other trusted party if the "delegatable" attribute is set to be "yes". In order for the client to validate the received message, each editor must leave a modification record in the corresponding segment. These modification records indicate when, how and by whom the content was modified. The modification records are secured by the editor's signatures.

Figure 3 is the Document Type Definition (DTD) of DISM. In the following sections, we will discuss the DISM elements with examples. Some of these examples may be represented in an informal form, in which some attributes and details are omitted, so that it looks shorter and clearer. We will highlight the names of the elements and attributes with " " the first time they appear in this section.

```

<?xml version="1.0"?>
<!DOCTYPE DISM SYSTEM "DISM.dtd">

<!-- ELEMENT manifest (publisher, URI, segment+, index) -->
<!-- ELEMENT publisher (#PCDATA) -->
<!-- ELEMENT URI (#PCDATA) -->
<!-- ELEMENT index segment-ID+, Signature -->
<!-- ATTLIST index
      randomizer CDATA #REQUIRED
      validity      ("valid","invalid") "valid">

<!-- ELEMENT segment      (segment-ID, content, permission*, modification) -->
<!-- ATTLIST segment
      id ID #REQUIRED
      randomizer CDATA #REQUIRED
      data-type CDATA #REQUIRED
      cachable ("yes"|"no") "no"
      validity ("valid","invalid") "valid"
      time-to-live CDATA #REQUIRED>

<!-- the ID that uniquely identifies the segment -->
<!-- ELEMENT segment-ID EMPTY -->
<!-- ATTLIST segment-ID
      id ID #REQUIRED
      content-digest CDATA #REQUIRED
      URI CDATA #REQUIRED
      if-invalid
      byte-range CDATA #IMPLIED>

<!-- ELEMENT content (choose | #PCDATA) -->
<!-- ELEMENT choose (when*, otherwise) -->
<!-- ELEMENT when content -->
<!-- ELEMENT otherwise content -->
<!-- ATTLIST when test CDATA #REQUIRED -->

<!-- ELEMENT permission (authorizer, source-permission?, editors, conditions, action, Signature) -->
<!-- ELEMENT authorizer (#PCDATA) -->
<!-- ELEMENT source-permission (permission) -->
<!-- ELEMENT editors (editor+) -->
<!-- ELEMENT editor (#PCDATA) -->
<!-- the definition of condition is not complete yet -->
<!-- ELEMENT conditions (condition+) -->
<!-- ELEMENT condition EMPTY -->
<!-- ATTLIST condition
      default ("true"|"false") #IMPLIED
      user-group CDATA #IMPLIED
      bandwidth CDATA #IMPLIED
      time-passed CDATA #IMPLIED>

<!-- ELEMENT action EMPTY -->

<!-- ATTLIST permission
      id ID #REQUIRED
      delegatable ("yes"|"no") "no">

<!-- ATTLIST action
      id ID #REQUIRED
      default-method
      ("Publish"|"Remove"|"Replace"|"Insert"|"Transcode"|"Alternative") #IMPLIED
      extended-method CDATA #IMPLIED
      max-size CDATA #IMPLIED
      encoding CDATA #IMPLIED
      choice CDATA #IMPLIED>

<!-- ELEMENT modification (previous-modification?, permission?, delegate, old-segment-ID?, new-
segment-ID, Signature) -->
<!-- ELEMENT previous-modification (modification) -->
<!-- ELEMENT old-segment-ID (segment-ID) -->
<!-- ELEMENT new-segment-ID (segment-ID) -->

```

Figure 3: DISM DTD

4.1. Manifest

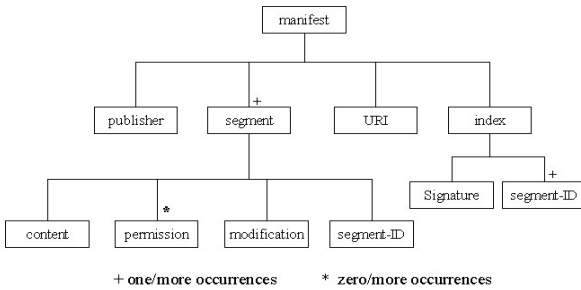


Figure 4: Content of the "manifest" Element

The "manifest" element is the root element a DISM message. A manifest must include the identity of the content publisher — a "publisher" element, the URI of the content — a "URI" element, the structure of the content — an "index" element, and at least one "segment" element. (Figure 4) The following is a simple example of a manifest.

```
<DISM:manifest>
<DISM:publisher> www.helloworld.com
</DISM:publisher>
<DISM:URI> www.helloworld.com/index.html
</DISM:URI>

<DISM:segment>
<DISM:segment-ID ID="001" content-digest="aaa"/>
<DISM:content> Hello world! </DISM:content>
<DISM:modification> ... </DISM:modification>
</DISM:segment>

<DISM:segment>
<DISM:segment-ID ID="002" content-digest="xxx"/>
<DISM:content> Hello world again! </DISM:content>
<DISM:modification> ... </DISM:modification>
</DISM:segment>

<DISM:index>
<DISM:segment-ID ID="001" content-hash="aaa"/>
<DISM:segment-ID ID="002" content-hash="xxx"/>
<DISM:Signature signatureValue="2346225">
</DISM:index>
</DISM:manifest>
```

Figure 5: Example of DISM "manifest"

Figure 5 is a manifest published by www.helloworld.com. The delivered content is www.helloworld.com/index.html. The file is divided into two segments and no modification permission is given. The "index" element lists the ID of the segments. A signature is put on the index to prevent the segments from been removed.

It is important to note that segments are not allowed to be removed, not even by the editor who is authorized to remove part of the content. Then how does the editor execute the "remove" permission? The editor can remove the "content" element inside the segment. The purpose of this rule is to avoid validation error. If the

whole segment is removed, the permissions and modification records of that segment will all be lost, causing validation failure. For the same reason, an editor cannot directly insert any segment, but he can insert contents into a blank content element.

4.2. Segment

```
<DISM:segment randomizer="...">
<DISM:content> Hello world! </DISM:content>

<DISM:permission ID="003">
<DISM:authorizer> proxy1 </DISM:authorizer>
<DISM:editors>
<DISM:editor> proxy3 </DISM:editor>
<DISM:editors>
<DISM:conditions>
<DISM:condition bandwidth="<256bps"/>
<DISM:conditions>
<DISM:action type="Remove"/>
<DISM:Signature> 67890 </DISM:Signature>
</DISM:permission>

<DISM:modification>
<DISM:permission ID="002">
<DISM:authorizer> www.helloworld.com
</DISM:authorizer>
<DISM:editors>
<DISM:editor> proxy2 </DISM:editor>
<DISM:editors>
<DISM:conditions>
<DISM:condition user-group="Robert"/>
<DISM:conditions>
<DISM:action type="Replace">
<DISM:Signature> 23456 </DISM:Signature>
</DISM:permission>
<DISM:previous-modification> . . .
</DISM:previous-modification>
<DISM:editor> proxy2 </DISM:editor>

<DISM:old-segment-ID>
<DISM:segment-ID ID="001" content-
digest="aaa"/>
</DISM:old-segment-ID>
<DISM:new-segment-ID>
<DISM:segment-ID ID="001" content-
digest="yy"/>
</DISM:new-segment-ID>

<DISM:Signature> 45678 </DISM:Signature>
</DISM:modification>

<DISM:segment-ID ID="001" content-
digest="yyy"/>
</DISM:segment>
```

Figure 6: Example of DISM "segment"

The segment element contains not only the page content but also its own permission set, modification record. Each element must contain a "segment-ID" element and a "modification" element (the modification record). A segment may also optionally contain a content element and any number of permission elements. The segment-ID element unambiguously identifies its parent segment – the segment that contains the segment-ID. The modification element records every change made to the segment. The permission elements

specify how the segment should be modified, and by whom. The content element stores the actual content of the web page. The following is a simple example of a segment. Figure 6 is a segment, which is originally part of `www.helloworld.com/index.html`. The segment is given two modification permissions. Permission "003" allows the editor to remove this part of the content when the network bandwidth is lower than 256 bps. Permission "002", which has been executed by proxy 2, allows the editor to replace the content if the user is "Robert". The execution of permission "002" is recorded in the modification element.

One of the effective methods to crack the public-key security system is to have the private key holder to encrypt deliberately selected string. This is a serious problem to DISM because all the contents are protected by the server's signature and some of these contents may come from unknown parties. To avoid this problem, we introduce an attribute named "randomizer" to each segment. A randomizer is an empty element containing a random string of any length (less than 128 bytes). Like the comments in the program code, the randomizer has no effect on the delivered content, but its existence makes the digest value of the segment different and unpredictable. The use of randomizer prevents the key holder's private key from being easily stolen.

4.2.1. Segment-ID

The segment-ID element is the identifier of its parent segment element – the segment that contains the segment-ID. The segment-ID is an empty element. It MUST include an ID of the parent segment, the content-digest of its parent segment and the canonical-function used to generate the content-digest. A segment-ID uniquely identifies its parent segment by presenting the content-digest. It is required that the canonical-function be complex enough so that it is computationally infeasible to forge a different copy of meaningful content which has the same digest value.

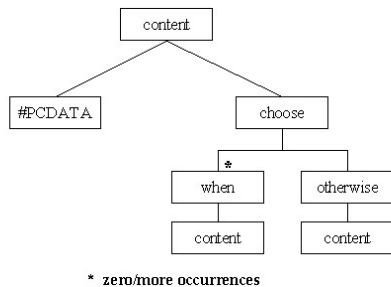


Figure 7: Content of "content" Element

4.2.2. Content

A content element may contain either the actual content of the web page or a conditional structure — choose|when|otherwise (see Figure 7). Recursion is prohibited. A content element, which is already enclosed inside a "choose" element, cannot further contain any other "choose" element.

4.2.3. Conditional Elements (choose|when|otherwise)

The conditional elements add the ability to perform logic based on expressions. All three must have an end tag. Every "choose" element must contain one "otherwise" element, and may optionally contain any number of "when" elements. Each when element or otherwise element must contain a content element. The following is an example of the conditional elements (in Figure 8).

```

<DISM:Content>
<DISM:choose>
  <DISM:when test="...">
    ...
  </DISM:when>
  <DISM:when test="...">
    ...
  </DISM:when>
  <DISM:otherwise>
    ...
  </DISM:otherwise>
</DISM:choose>
</DISM:Content>
  
```

Figure 8: Example of DISM "content"

DISM processors will execute the first when statement whose "test" attribute evaluates truthfully, and then exit the choose element. If no when element evaluates to true, and an otherwise element is present, that element's content will be executed.

Conditional elements use expressions (in their test attributes) to determine how to apply the contained elements. Expressions consist of operators, variables and literals, and evaluate to true or false. Single quotes are used within an expression to delimit literals. Whitespace is optional around all operators. The following set of unary and binary logical operators are supported by DISM expressions, listed in order of decreasing precedence:

Operator	Type
==, !=, <, >, <=, >=	Comparison
!	Unary negation
&	Logic and
	Logic or

Operands associate from left to right. Sub-expressions can be grouped with parentheses in order to explicitly specify association. If both operands are numeric, the expression is evaluated numerically. If either binary operand is non-numeric, both operands are evaluated as strings. After expansion, variables are loosely typed, but care should be taken. For example, a version reported as 3.01.23 or 1.05a will not test as a number.

The behavior of comparisons which incompatibly typed operators is undefined. If an operand is empty or undefined, the expression will always evaluate to false. Logical operators ("&", "|", "!") can be used to qualify expressions, but cannot be used as comparitors themselves.

4.3. Permission

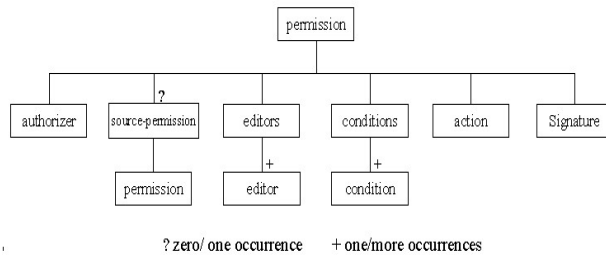


Figure 9: Content of "permission" Element

The server expresses its intentions in terms of permissions. Permissions specify how their parent segment can be modified. A permission element may contain the following elements (Figure 9):

1. An "authorizer" element (MUST): The authorizer element indicates the identity of the party who gives the permission. Normally, the authorizer of the permission is the content publisher himself. Nevertheless, if the permission is delegated, the authorizer should be the latest delegate.
2. A "source-permission" element (OPTIONAL): The source-permission element is optional to the permission element. A delegated permission needs to prove the legitimacy of its delegation. The source-permission does this job by providing the original permission from which the current permission is delegated.
3. An "editors" element (MUST): The editors element lists all the (at least one) editors who are eligible to execute the permission.
4. A "conditions" element (MUST): The conditions element lists all the (at least one) conditions, which, if evaluated truthfully, may trigger the execution of the permission. The listed conditions are connected by logic "OR".

5. An "action" element (MUST): The action element specifies how the content can be modified.
6. A "Signature" element (MUST)

The following (see Figure 10) is an example of a permission element.

```

<DISM:permission ID="003">
<DISM:authorizer> proxy1 </DISM:authorizer>

<DISM:source-permission>
<DISM:permission ID="001" delegatable="yes">
<DISM:authorizer> www.helloworld.com
</DISM:authorizer>
<DISM:editors>
<DISM:editor> proxy1 </DISM:editor>
<DISM:editor> proxy2 </DISM:editor>
</DISM:editors>
<DISM:conditions>
<DISM:condition user-group="Robert" />
</DISM:conditions>
<DISM:action type="replace" />
<DISM:Signature> 12345 </DISM:Signature>
</DISM:permission>
</DISM:source-permission>

<DISM:editors>
<DISM:editor> proxy3 </DISM:editor>
</DISM:editors>

<DISM:conditions>
<DISM:condition user-group="Robert" />
</DISM:conditions>

<DISM:action type="replace" encoding="UTF-8" />

<DISM:Signature> 67890 </DISM:Signature>
</DISM:permission>
  
```

Figure 10: Example of DISM "permission"

We will discuss the delegation and validation of the permission element in details in Section 6.

4.3.1. Conditions

The "conditions" element contains at least one "condition" element. If there are multiple condition elements, they are connected by logic "OR". In other words, the conditions element evaluates to be true if any of its condition elements is true. The attributes in a condition element are connected by logic "AND". Consider the following example in Figure 11:

```

<DISM:conditions>
<DISM:condition user-group="Robert" />
<DISM:condition bandwidth="<DISM:1.5kb/s" time-
passed=">123456" />
</DISM:conditions>
  
```

Figure 11: Example of DISM "content"

The example evaluates truthfully if either the user is "Robert" or the network bandwidth is lower than 1.5kb/s and web page is more than 123456 milliseconds

old. The condition is an open element. Users of DISM can introduce their own system attributes. However, a condition must be evaluated false if it contains any attribute that the processor does not understand.

4.3.2. Action

The "action" element specifies the method for modifying the segment. It is an empty element (i.e. there is no sub-element). The predefined method includes:

1. Publish: Initialising the content
2. Add: Adding a piece of content to an empty element.
3. Remove: Removing the content from an element (so that the element becomes empty).
4. Replace: Replacing the original content with new content
5. Transcode: Translating the content to another type of code or language
6. Alternate: Choosing the content among several available alternatives

It is important to note that the "Publish" action can only (and must) be done by the content provider and it does not require authorization.

4.4. Modification

The modification element records every action performed on its parent segment. It is intended for the use of validating the segment. A valid modification element may contain the following elements (Figure 12):

1. A "permission" element (OPTIONAL): The permission element shows the legitimacy of the modification. All modifications need permissions except one – publishing. The action "publish" does not require any authorization since it is performed by the content provider himself.
2. A "previous-modification" element (OPTIONAL): The previous-modification element contains the preceding modification records. It is the proof that the current modification is based on valid content. The first modification, which is always "publishing", has no previous-modification.
3. An "editor" element (MUST): The editor element shows the identity of the current editor.
4. An "old-segment-ID" element (MUST): This element refers to the content before the current modification.

5. A "new-segment-ID" element (MUST): This element refers to the content after the current modification.
6. A "Signature" element (MUST)

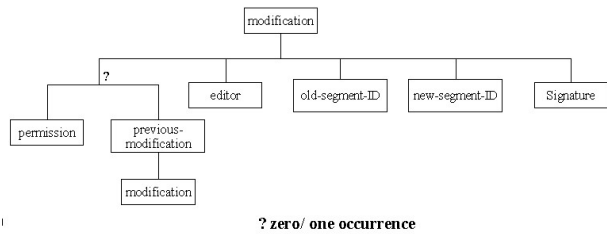


Figure 12: Content of "modification" Element

Figure 13 is an instance of a modification element.

```
<DISM:modification>
<DISM:permission ID=two>
  <DISM:authorizer> www.helloworld.com
</DISM:authorizer>
<DISM:editors>
  <DISM:editor> proxy2 </DISM:editor>
</DISM:editors>
  <DISM:conditions> ... </DISM:conditions>
  <DISM:action type="remove" />
  <DISM:Signature> 23456 </DISM:Signature>
</DISM:permission>

<DISM:previous-modification>
  <DISM:modification>
    <DISM:editor> www.helloworld.com
  </DISM:editor>
    <DISM:action type="publish" />
    <DISM:new-segment-ID>
      <DISM:segment-ID ID="002" content-digest="xxx" />
    </DISM:new-segment-ID>
    <DISM:Signature> 987654 </DISM:Signature>
  </DISM:modification>
</DISM:previous-modification>

<DISM:editor> proxy2 </DISM:editor>

<DISM:old-segment-ID>
<DISM:segment-ID ID="002" content-digest="xxx" />
</DISM:old-segment-ID>

<DISM:new-segment-ID>
<DISM:segment-ID ID="002" content-digest="yyy" />
</DISM:new-segment-ID>

<DISM:Signature> 45678 </DISM:Signature>
</DISM:Modification>
```

Figure 13: Example of DISM "modification"

In Figure 13, the editor making the modification is proxy2. From the permission element, we know that proxy2 is authorized by www.helloworld.com. The previous-modification tells us that the content was published by www.helloworld.com. It is important to note that the "old-segment-ID" of current modification must match the new-segment-ID of the previous-

modification. Through this way, the modification records are linked together, making local validation possible. The modification element plays a very important role in the message validation. We will discuss the details of message validation, authorization and validation in Section 6.

4.5. Index

In DISM, deleting, inserting or reordering of segments are strictly prohibited mainly for the purpose to avoid validation failure. However, these actions are hard to detect because segments are independent of each other. The "index" element is intended for use to guard the segments. An index element must contain exactly one signature element and at least one segment-ID element.

The index element is supposed to contain the segment-IDs of all the segments in the manifest. Each segment-ID unambiguously identifies a segment. The index element must also enclose the latest editor's signature. The editor, after making modification and updating the modification record, must also update the segment-ID in the index element and leave his signature in the index element.

5. Authorization and Validation Model

DISM allows the content owner to express their intention, in terms of permissions, on how the page content should be presented; it allows the proxies to execute or delegate the server's permissions; and it allows any party to validate the page content with respect to the server's permissions.

Each modification made to the content must be unambiguously recorded and bound to the editor's identity by his signature. It is these signatures, through which we can validate the message without requesting the original server. If the editor's signature is missing or the content and the signature do not match, the corresponding segment must be considered dubious.

5.1. Authorization

The server is the only one who has the privilege to make authorization. Through authorization, the server specifies the policy on how the content should be presented. Firstly, the server will divide the content into segments. Secondly, the server will specify the permis-

sions for each modifiable segment. Lastly, the server will collect all the segment-IDs to make an index element and attach it to the end of the manifest. Both the index and the permissions must to be signed. Consider the following example in Figure 14:

```
<DISM:manifest>
<DISM:publisher> www.helloworld.com
</DISM:publisher>
<DISM:URI> www.helloworld.com/index.html
</DISM:URI>

<DISM:segment>
<DISM:content> Hello world! </DISM:content>
<DISM:permission ID="001" delegatable="yes">
<DISM:authorizer> www.helloworld.com
</DISM:authorizer>
<DISM:editors>
<DISM:editor> proxy1 </DISM:editor>
</DISM:editors>
<DISM:conditions>
<DISM:condition user-group="Robert"/>
</DISM:conditions>
<DISM:action type="replace"/>
<DISM:Signature> 12345 </DISM:Signature>
</DISM:permission>
<DISM:modification> . . . </DISM:modification>
<DISM:segment-ID ID="001" content-digest="aaa"/>
</DISM:segment>

<DISM:segment>
<DISM:content> Hello world again!
</DISM:content>
<DISM:permission ID="002">
<DISM:authorizer> www.helloworld.com
</DISM:authorizer>
<DISM:editors>
<DISM:editor> proxy2 </DISM:editor>
</DISM:editors>
<DISM:conditions>
<DISM:condition bandwidth="<DISM:1kb/s">
</DISM:conditions>
<DISM:action type="remove"/>
<DISM:Signature> 23456 </DISM:Signature>
</DISM:permission>
<DISM:modification> . . . </DISM:modification>
<DISM:segment-ID ID="002" content-
digest="xxx"/>
</DISM:segment>

<DISM:index>
<DISM:segment-ID ID="001" content-
digest="aaa"/>
<DISM:segment-ID ID="002" content-
digest="xxx"/>
<DISM:Signature> 2346225 </DISM:Signature>
</DISM:index>
</DISM:manifest>
```

Figure 14: Example of DISM Message with "permissions"

The above example is a manifest of the web page www.helloworld.com/index.html. The manifest consists of two segments each of which contains its own permission. Permission one is given to proxy1 allowing it to "replace" the content of segment one in case the user is "Robert". Permission "002" is given to proxy2 allowing it to remove the content of segment "002" if the network bandwidth is lower than 1kb/s. Note that permission one

is specified to be "delegatable" which means that proxy1 is free to delegate the permission to any other party. Because the permissions are directly given by the publisher, they do not need to specify their source-permission.

5.2. Execution

```
<DISM:manifest>
<DISM:publisher> www.helloworld.com
</DISM:publisher>
<DISM:URI> www.helloworld.com/index.html
</DISM:URI>

<DISM:segment>
...
<DISM:segment-ID ID="001" content-
digest="aaa"/>
</DISM:segment>

<DISM:segment>
<DISM:content></DISM:content>

<DISM:modification>
<DISM:permission ID="002">
<DISM:authorizer> www.helloworld.com
</DISM:authorizer>
<DISM:editors>
<DISM:editor> proxy2 </DISM:editor>
</DISM:editors>
<DISM:conditions>
<DISM:condition bandwidth="<DISM:lkb/s">
</DISM:conditions>
<DISM:action type="remove"/>
<DISM:Signature> 23456 </DISM:Signature>
</DISM:permission>

<DISM:editor> proxy2 </DISM:editor>

<DISM:old-segment-ID>
<DISM:segment-ID ID="002 content-
digest="xxx"/>
</DISM:old-segment-ID>

<DISM:new-segment-ID>
<DISM:segment-ID ID="002" content-
digest="yyy"/>
</DISM:new-segment-ID>

<DISM:Signature> 45678 </DISM:Signature>
</DISM:modification>

<DISM:segment-ID ID="002" content-
digest="yyy"/>
</DISM:segment>

<DISM:index>
<DISM:segment-ID ID="001" content-
digest="aaa"/>
<DISM:segment-ID ID="002" content-
digest="yyy"/>
<DISM:Signature> 237890 </DISM:Signature>
</DISM:index>

</DISM:manifest>
```

Figure 15: Example of DISM Message after Permission Execution

Execution here refers to the execution of the permissions. A permission can be executed if the following conditions are satisfied:

1. The permission is valid.
2. The parent segment of the permission is valid.
3. The conditions of the permission evaluate to be true.
4. The proxy is authorized to execute the permission.

In the execution a proper editor must:

- 1) Perform the action specified by the permission.
- 2) Remove the permission from the manifest.
- 3) Update the modification record and sign it,
- 4) Update the <DISM:index> element and sign it.

Continuing the example in Figure 14, after executing permission "002", the DISM message becomes the instance in Figure 15:

In the above example, permission "002" is executed by the proxy2. Compare the manifest with its original form in Figure 14. Three changes are made to the manifest. Firstly, a modification element is created. Secondly, the body of permission "002" is migrated into the modification element. Lastly, the index element is updated and signed. Note that the authorized action of permission "002" is "remove". However, the editor is not allowed to remove the entire segment. Only the content element can be removed. The removal of a whole segment will cause the loss of its modification records and leads to validation failure.

5.3. Delegation

A permission can be delegated if the following conditions are satisfied:

1. The permission is valid
2. The parent segment of the permission is valid.
3. The proxy is authorized by the permission.
4. The permission is specified to be "delegatable".

A proper editor must do the following things in the delegation:

- 1) Insert a new permission
- 2) Migrate the old permission into the source-permission element of the new permission.

Continuing the example in Figure 14, after the permission "001" being delegated by the proxy1, the manifest becomes the instance in Figure 16. Permission "003" is the delegated copy of permission "001" in Figure 14. In DISM, a delegated permission must state the source of its authority, which is the instance of the permission from which the current permission is delegated. In this

example, The ID of the new permission is "003". It encompasses the permission "001" as its source-permission.

```
<DISM:manifest>
<DISM:manifest>
<DISM:publisher> www.helloworld.com
</DISM:publisher>
<DISM:URI> www.helloworld.com/index.html
</DISM:URI>

<DISM:segment>
  <DISM:content> Hello world! </DISM:content>

  <DISM:permission ID="003">
    <DISM:authorizer> proxy1 </DISM:authorizer>
    <DISM:source-permission>
      <DISM:permission ID="001" delegatble="yes">
        <DISM:authorizer> www.helloworld.com
      </DISM:authorizer>
    <DISM:editors>
      <DISM:editor> proxy1 </DISM:editor>
    </DISM:editors>
    <DISM:conditions>
      <DISM:condition user-group="Robert"/>
    </DISM:conditions>
    <DISM:action type="replace"/>
    <DISM:Signature> 12345 </DISM:Signature>
  </DISM:permission>
  <DISM:source-permission>
    <DISM:editors>
      <DISM:editor> proxy3 </DISM:editor>
    </DISM:editors>
    <DISM:conditions>
      <DISM:condition user-group="Robert"/>
    </DISM:conditions>
    <DISM:action type="replace"/>
    <DISM:Signature> 67890 </DISM:Signature>
  </DISM:permission>

  <DISM:modification> . . .
</DISM:modification>

  <DISM:segment-ID ID="001" content-
  digest="aaa"/>
</DISM:segment>

<DISM:segment>
. . .
<DISM:segment-ID ID="002" content-
digest="xxx"/>
</DISM:segment>

<DISM:index>
<DISM:segment-ID ID="001" content-
digest="aaa"/>
<DISM:segment-ID ID="002" content-
digest="xxx"/>
<DISM:Signature> 2346225 </DISM:Signature>
</DISM:index>
</DISM:manifest>
```

Figure 16: Example of DISM Message after Permission Delegation

5.4. Validation

A manifest is valid if the all of the following conditions are satisfied:

1. All of its segments are valid.
2. All of its segments are in the correct order.

The condition 1 is checked by validating the permissions and the modification records. The condition 2 is checked by validating the index element.

5.4.1. Validating a Permission

A permission element is valid if the following conditions are satisfied:

1. The source-permission, if any, is valid and it is delegatable.
2. The authorizer is either the publisher of the manifest or a member of the editors of the source-permission.
3. The signature is valid and it is provided by the authorizer.

Here is an example of an invalid permission:

```
<DISM:permission ID="003">
  <DISM:authorizer> proxy4 </DISM:authorizer>

  <DISM:source-permission>
    <DISM:permission ID="001">
      <DISM:authorizer> www.helloworld.com
    </DISM:authorizer>
    <DISM:editors>
      <DISM:editor> proxy1 </DISM:editor>
      <DISM:editor> proxy2 </DISM:editor>
    </DISM:editors>
    <DISM:conditions>
      <DISM:condition user-group="Robert"/>
    </DISM:conditions>
    <DISM:action type="replace"/>
    <DISM:Signature> 12345 </DISM:Signature>
  </DISM:permission>
  <DISM:source-permission>

  <DISM:editors>
    <DISM:editor> proxy3 </DISM:editor>
  </DISM:editors>

  <DISM:conditions>
    <DISM:condition user-group="Robert"/>
  </DISM:conditions>

  <DISM:action type="replace"/>

  <DISM:Signature> 67890 </DISM:Signature>
</DISM:permission>
```

Figure 17: Example of Invalid Permission

Figure 17 is invalid in two places (underlined). Firstly, the source-permission is not "delegatable". Unless explicitly declared, all the permissions are not "delegable" by default. Secondly, the authorizer of the new permission, which is proxy4, is not a member of the editors of the source-permission. Validating the digital signature is done by decrypting the signature value and comparing the result with the content digest. It is not in the scope of this paper.

5.4.2. Validating a Modification

A modification record is valid if the following conditions are satisfied:

1. The new-segment-ID refers to the parent segment correctly.
2. The permission element is valid.
3. The editor is one of the editors authorized by the permission.
4. The modification performed is consistent with the permitted action. E.g. If the permitted action is "Remove", the content element must be empty.
5. If the editor is not the server himself, the old-segment-ID element and the previous-modification element must present. The old-segment-ID must match the new-segment-ID in the previous-modification element.
6. The previous-modification, if any, is valid.
7. The signature is valid and it is provided by the editor.

Here is an example of an invalid modification element (see Figure 18):

```
<DISM:modification>
<DISM:permission ID="002">
<DISM:authorizer> www.helloworld.com
</DISM:authorizer>
<DISM:editors>
<DISM:editor> proxy2 </DISM:editor>
</DISM:editors>
<DISM:conditions>
<DISM:condition bandwidth="<DISM:lkb/s"/>
</DISM:conditions>
<DISM:action type="remove"/>
<DISM:Signature> 23456 </DISM:Signature>
</DISM:permission>

<DISM:previous-modification>
<DISM:modification>
<DISM:editor> www.helloworld.com </DISM:editor>
<DISM:action type="Publish"/>
<DISM:new-segment-ID>
<DISM:segment-ID ID="002" content-
digest="xxx"/>
</DISM:new-segment-ID>
<DISM:Signature> 987654 </DISM:Signature>
</DISM:modification>
</DISM:previous-modification>

<DISM:editor> proxy4 </DISM:editor>

<DISM:old-segment-ID>
<DISM:segment-ID ID="002" content-digest="aaa">
</DISM:old-segment-ID>

<DISM:new-segment-ID>
<DISM:segment-ID ID="002" content-digest="yyy">
</DISM:new-segment-ID>

<DISM:Signature> 45678 </DISM:Signature>
</DISM:Modification>
```

Figure 18: Example of Valid Modification Record

Figure 18 is invalid in two places (underlined). Firstly, the editor, which is proxy4, is not a member of the authorized editors of permission "002". Secondly, the old-segment-ID does not match the new-segment-ID in the previous-modification. This indicates that the content of the segment may have been tampered between the previous modification and the current modification.

5.4.3. Validating an Index

An index element is valid if the following conditions are satisfied:

1. There is a one-to-one mapping between the segments in the manifest and the segment-IDs in the index.
2. The segment-IDs and their matching segments are in the same order.
3. The signature in the index element is valid and it is provided by one of the authorized editors.

The following is an example of an invalid "index" element:

```
<DISM:manifest>
<DISM:manifest>
<DISM:publisher> www.helloworld.com
</DISM:publisher>
<DISM:URI> www.helloworld.com/index.html
</DISM:URI>

<DISM:segment>
...
<DISM:segment-ID ID="001" content-
digest="aaa"/>
</DISM:segment>

<DISM:segment>
...
<DISM:segment-ID ID="002" content-
digest="xxx"/>
</DISM:segment>

<DISM:segment>
...
<DISM:segment-ID ID="003" content-
digest="yyy"/>
</DISM:segment>

<DISM:index>
<DISM:segment-ID ID="003" content-
digest="yyy"/>
<DISM:segment-ID ID="002" content-
digest="xxx"/>
<DISM:Signature> 2346225 </DISM:Signature>
</DISM:index>
</DISM:manifest>
```

Figure 19: Example of DISM Message after Permission Delegation

In Figure 19, the "index" element is invalid because:

1. The ID of the segment "001" does not appear in the index. This indicates that segment "001" may be inserted by some unauthorized party.
2. The positions of segment "003" and "002" are exchanged.

5.4.4. Actions for Validation Failure

If the validation fails, the receiving party have two options:

1. Discard the invalid message and make a reply of failure (either HTTP 404 "page not available" or HTTP 500 "server error");
2. Mark the invalid part and forward it. Both the "segment" element and the "index" element have an attribute named "validity", whose default value is "valid". If the validation fails, this attribute can be set to "invalid", warning the following receivers.

The editor SHOULD not do further modification or delegation after finding the content to be invalid.

6. The Cache Model

DISM is friendly to caches because the actual content is separated from the permissions and modification records. Caches can keep the segments and as separate items. The segment-IDs uniquely identify each segment. The following is the definition of the "invalidation.dtd".

Invalidation of DISM messages is done by posting an invalidation form to the server. This is similar to the invalidation DTD of the Edge Side Include (ESI). The following is the definition of the "invalidation.dtd":

```

1. <!ELEMENT invalidation (segment-ID+,
   system-info)>
2. <!ATTLIST invalidation URI CDATA #REQUIRED>

3. <!ELEMENT system-info EMPTY>
4. <!ATTLIST system-info
5.     time-passed CDATA #IMPLIED
6.     user-group CDATA #IMPLIED
7. >

8. <!ELEMENT invalidation-response (segment*)>
9. <!ATTLIST invalidation-response URI CDATA
   #REQUIRED>

```

Figure 20: Invalidation DTD

In Figure 20, lines 1~7 define the format of the invalidation request. Lines 8 and 9 define the format of the invalidation-response. The body of the invalidation request is a valid XML document, in which a list of one

quest is a valid XML document, in which a list of one or more segment-IDs is given. The invalidation element also contains a "system-info" element. The system-info is intended for use to send system information such as regional settings or user details. Consider the following example in Figure 21:

```

POST /dism-invalidate HTTP/1.0
Authorization: Basic
aW52YWxpZGF0b3I6aW52YWxpZGF0b3I=
Content-Length: 217

<?xml version="1.0" ?>
<!DOCTYPE invalidation SYSTEM
"invalidation.dtd">
<DISM:invalidation
URI="www.hollowworld.com/index.html">
<DISM:segment-ID ID="001" content-hash="aaa"/>
<DISM:segment-ID ID="002" content-hash="xxx"/>
<DISM:system-info time-passed="2172734" user-
group="Kate"/>
</DISM:invalidation>

```

Figure 21: Example of Invalidation Request

DISM invalidation request is submitted by using HTTP/1.0 POST method. In Figure 21, the client request for two segments "001" and "002". (In the real world, the two segment-IDs should contain URI so that the segment can be located. We ignore it in this paper to make the examples shorter.) The system-info element tells the status of the client – the user is "Kate" and the last time refreshing the cache was 2172734 milliseconds ago.

```

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 224

<?xml version="1.0"?>
<!DOCTYPE invalidation-response SYSTEM
"invalidation.dtd">
<DISM:invalidation-response
URI="www.helloworld.com/index.html">
<DISM:segment>
<DISM:content content-modified=yes>"Hello
Kate!"</DISM:content>
...
</DISM:segment>
</DISM:segment>
<DISM:content content-
modified="no"></DISM:content>
...
</DISM:segment>
</DISM:invalidation-response>

```

Figure 22: Example of Invalidation Response

The invalidation-response consists of a list of content elements. Each content element corresponds to a segment-ID listed the invalidation request. If a segment remains unchanged, an empty content element must be returned with the attribute content-modified set to "no". Otherwise, the content element must carry the new content of the segment with its content-modified attribute

set to "yes". Two responses fail the invalidation request: 1) Any non-200 HTTP response code returned by the server; 2) An invalidation-response missing some segments. Here is an example of a successful invalidation (see Figure 22).

Since the HTTP GET method is incapable of sending complex cache invalidation message, DISM invalidation submits its invalidation request by using HTTP/1.0 POST method. Every DISM supported proxy along the path should strip off the segment-IDs that they are capable to update and submit the rest to the next proxy. When the last segment is updated, the invalidation request completes successfully. If even the server fails to update some of the segment, the invalidation request fails and a non-200 HTTP response code must be returned.

7. Comparing DISM with SSL

DISM is not a simple substitution of SSL in the field of data integrity. The Secure Socket Layer (SSL) technology is a generic solution to protecting all sorts of web communications. SSL seeks to keep the communication private and confidential by encrypting every bit in the data stream. In contrast, DISM aims to protect open messages from being tampered and allow various value-added services.

The nature of absolute confidentiality of SSL makes it unsuitable for the server-proxy-client infrastructure. SSL secures the data stream from the TCP/IP layer, strictly prohibiting any third party to intercept the communication. However, this also disables many valuable services like caching, content customisation, request routing etc. DISM is intended to address both data integrity and the web services. DISM allows proxies to modify the web content as long as they can prove that they are authorized to do so. DISM also allows the content receiver to validate the content with respect to the server's intentions.

Another advantage of DISM over SSL is pre-processing. Since only part of the content is to be modified by the proxies, the server has the freedom to pre-process the static parts. Actually, even the dynamic parts can be pre-processed if their editor groups and modification permissions are relatively static. Pre-processing helps the server to balance the workload between the peak hours and off-peak hours.

Comparing the performance of DISM and HTTPS – the HTTP extension of SSL – may not be very meaningful since they are designed to address different problems. Yet, in one simple but common case, in which no value-added service is available, the two models are comparable. Our experiment shows that, even in this case, DISM outperforms HTTPS.

The experiment is carried out in a high-speed local area network (LAN) whose estimated round trip time is less than 10 ms, and the average bandwidth is more than 50 MB/s. The server and the proxy resides on the same SUN SparcServer 1000 with 8 processors, and the client is a PC with Windows XP and Pentium III 500 CPU.

Table 1 and 2 shows the performance of HTTPS and DISM on the server side correspondingly. Two sets of data are collected: the server response time and the size of the response message. We send HTTPS request from a nearby client and collect data from the log file of the intermediary proxy running Squid [8]. Since the server, the proxy and the client reside on the same LAN, the effect of network transmission is negligible. The test of DISM is carried out in the same network through HTTP. The proxy's cache function is disabled. Since DISM does not require extra encryption overhead (because of pre-processing), it is reasonable to assume that serving a DISM message is equivalent to serving "the HTTP message + the DISM tags". We add 8 Kbytes (estimated average DISM tag size for a single, non-segmentized file) to each sample file and test them through HTTP protocol.

No. of Run	0.5 MB		1 MB		2 MB		4 MB		8 MB	
	Time (ms)	Size (kb)	Time (ms)	Size (kb)	Time (ms)	Size (kb)	Time (ms)	Size (kb)	Time (ms)	Size (kb)
1	137	498	193	10177	371	2034	1228	4206	2268	8412
2	122	498	198	10177	439	2034	1027	4206	3183	8412
3	137	498	213	10177	421	2034	1200	4206	2932	8412
Average	132	498	201	10177	410	2034	1151	4206	2794	8412

Table 1: Performance of HTTPS

No. of Run	0.5 MB		1 MB		2 MB		4 MB		8 MB	
	Time (ms)	Size (kb)	Time (ms)	Size (kb)	Time (ms)	Size (kb)	Time (ms)	Size (kb)	Time (ms)	Size (kb)
1	66	487	129	1024	254	2041	611	4205	1251	8400
2	59	487	116	1024	232	2041	597	4205	1214	8400
3	61	487	118	1024	229	2041	592	4205	1233	8400
Average	62	487	121	1024	238	2041	600	4205	1232	8400

Table 2: Performance of DISM

Table 1 and 2 shows the performance of HTTPS and DISM on the server side correspondingly. Two sets of data are collected: the server response time and the size of the response message. We send HTTPS request from a nearby client and collect data from the log file of the intermediary proxy running Squid [8]. Since the server, the proxy and the client reside on the same LAN, the effect of network transmission is negligible. The test of DISM is carried out in the same network through HTTP. The proxy's cache function is disabled. Since DISM does not require extra encryption overhead (because of pre-processing), it is reasonable to assume that serving a DISM message is equivalent to serving "the HTTP message + the DISM tags". We add 8 Kbytes (estimated average DISM tag size for a single, non-segmentized file) to each sample file and test them through HTTP protocol.

The result indicates that both HTTPS and DISM add very little extra bytes to the original content. However, HTTPS' server response speed is much lower than the speed of DISM due to the encryption overhead. Figure 23 visualizes this difference. The server response time of HTTPS is approximately 2 times of the response time of DISM, and this ratio tends to go bigger when the size of the sample file increases. And the performance of pre-processed DISM is close to that of normal HTTP.

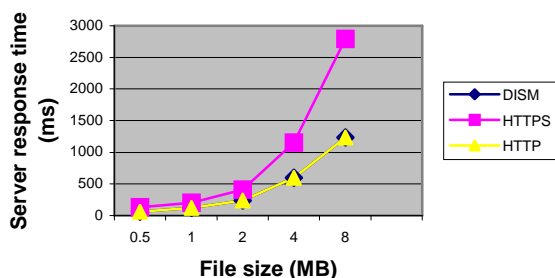


Figure 23: Performance of DISM vs. HTTPS

8. Conclusion and Future work

In this paper, we proposed an XML-based model, named DISM, to address data integrity of web contents.

DISM is an XML language designed to be interposed into markup to preserve data integrity of the web contents. It allows the content owner to express their permissions as part of the response message; it allows the proxies to execute and delegate the owner's intentions; and it allows any party to validate the page content with respect to the owner's permissions. We provided the DTD of DISM and its specifications. We have also developed a set of rules for cache and cache invalidation for DISM messages. There are several directions to focus on to improve this model. One of the immediate needs is to refine the parameters of the permission and its sub-elements, especially the details of the actions. Extending the model to address data integrity of request messages is also very important.

References:

- [1] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia and Ed Simon, "XML-Signature Syntax and Processing", W3C Recommendation 12 February 2002
- [2] Jeremy Elson and Alberto Cerpa, "ICAP the Internet Content Adaptation Protocol", The ICAP protocol group, June 2001
- [3] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte and Dave Winer, "Simple Object Access Protocol (SOAP) 1.1", W3C Note 08 May 2000
- [4] Erik Christensen, Francisco Curbera, Greg Meredith and Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1", W3C Note 15 March 2001
- [5] R. Fielding, J. Gettys, J. Mogual, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol-HTTP/1.1", RFC 2616, June 1999
- [6] <http://home.netscape.com/security/techbriefs/ssl.html>
- [7] Hilarie K. Orman, Volera, Inc. "Data Integrity for Mildly Active Content", Active Middleware Workshop, August 14, 2001
- [8] Andres Kroonmaa "Squid refresh and LRU logic. Proposal", MicroLink Online, 15-Jan-1997