11-2003

AA Flexible and Scalable Authentication Scheme for JPEG 2000 Image Codestreams

Cheng PENG
Institute for Infocomm Research

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Yongdong WU
Institute for Infocomm Research

Weizhong SHAO
Peking University

DOI: https://doi.org/10.1145/957013.957101

Citation

PENG, Cheng; DENG, Robert H.; WU, Yongdong; and SHAO, Weizhong. AA Flexible and Scalable Authentication Scheme for JPEG 2000 Image Codestreams. (2003). *MM'03: Proceedings of the 11th ACM International Conference on Multimedia, Berkeley, CA, November 4-6, 2003*. 441-443. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/1076

A Flexible and Scalable Authentication Scheme for JPEG2000 Image Codestreams

Cheng Peng[1,2], Robert Deng[1], Yongdong Wu[1]
[1] Institute for Infocomm Research
Singapore 119613
{pengcheng, deng, wydong}
@i2r.a-star.edu.sg

Weizhong Shao[2]
[2] Department of Computer Science and Technology
Peking University, Peking
China 100871
wzshao@pku.edu.cn

ABSTRACT

JPEG2000 is an emerging standard for still image compression and is becoming the solution of choice for many digital imaging fields and applications. An important aspect of JPEG2000 is its "compress once, decompress many ways" property [1], i. e., it allows extraction of various sub-images (e.g., images with various resolutions, pixel fidelities, tiles and components) all from a single compressed image codestream. In this paper, we present a flexible and scalable authentication scheme for JPEG2000 images based on the Merkle hash tree and digital signature. Our scheme is fully compatible with JPEG2000 and possesses a "sign once, verify many ways" property. That is, it allows users to verify the authenticity and integrity of different sub-images extracted from a single compressed codestream protected with a single digital signature.

Categories and Subject Descriptors

K.4.4 [**Computers and Society**]: Electronic Commerce—*intellectual property, security*; I.3.8 [**Computer Methodologies**]: Computer Graphics—*applications*.

General Terms

Security

Keywords

JPEG2000, Authentication, Digital signature, One-way hash function, Message digest, Merkle hash tree, Data integrity, Image compression.

1. INTRODUCTION

JPEG2000 [1, 2, 3] is a wavelet based emerging standard for still image compression and is fast becoming the solution of choice for many digital imaging fields and applications. As digital imagery becomes more and more pervasive in our lives, image compression must not only reduce the amount of data storage and transmission bandwidth requirements, but also allow extraction for processing and targeting for different devices and applications. JPEG2000 achieves state-of-the-art low bit rate compression performance and has a rate distortion advantage over the original JPEG. Moreover, it is designed for "compress once, decompress many ways" [1], i. e., it supports extraction of different versions of an image (e. g., different resolutions, pixel fidelities, components, and so on), all from a single compressed image codestream. This allows an application to manipulate or transmit only the essential image data for any target device from any JPEG2000 compressed codestream. In this paper, we refer to images, with various resolution levels, pixel fidelities, components and so on, extracted from a single image codestream as the $sub-images$ of the codestream.

Digital imagery is playing an increasingly important role in sensitive application fields such as government, finance, health care and the law. In application fields such as these, it is critical and often a requirement for recipients to check the integrity of the received image data as well as the authenticity of the origin of the image. Many data integrity and origin authentication techniques have been proposed [4, 5]. However, these generic data authentication techniques completely ignore the internal data structure of the content under protection. A scheme using digital signature for authenticating JPEG2000 codestreams is proposed in [6]: the scheme simply signs each code-block and attaches the digital signature to the end of the code-block bit stream. Hence, the scheme is neither secure nor efficient. It will generate many signatures since a codestream may contain many code-blocks. The scheme is vulnerable to cut-and-paste attack since it only authenticates individual code-blocks, not the image codestream as a whole.

Several semi-fragile JPEG2000 image authentication schemes are presented [7, 8, 9], which aim at authenticating images under lossy compression and other common image manipulations such as blurring and sharpening. For example, their schemes accept JPEG lossy compression on the watermarked image to a pre-determined quality factor, and reject malicious attacks. The objective of our authentication scheme is different from that of [7, 8, 9]: we aim at authenticating the sub-images transcoded from a single original image codestream. The sub-images have not only different qualities, but also different resolutions, components and spacial regions.

In this paper, we introduce a flexible and scalable authentication scheme for JPEG2000 image codestreams based on the Merkle hash tree [10] and digital signature. Our scheme is fully compatible with JPEG2000 and possesses the so called "sign once, verify many ways" property. That is, it allows users to verify the authenticity and integrity of different sub-images transcoded from a single compressed codestream protected with a single digital signature. By doing so, our scheme preserves the important "compress once, decompress many ways" property of the JPEG2000 standard.

The concept of Merkle hash tree has been used for certifying answers to queries over XML documents [11], for proving the presence or absence of public key certificates on revocation lists [12, 13], and for certifying data published by untrusted publishers [14]. However, authenticating JPEG2000 image codestreams requires more careful treatment since these streams are not as structured and are subject to various progression orders which further complicates the issue. The innovative contribution of this paper is the development of a general authentication model of JPEG2000 codestreams using the Merkle hash tree which supports authentication of various sub-images with only one pre-computed digital signature and a small amount of auxiliary information. Another important contribution of the paper is the construction of the optimized Merkle hash tree which allows for minimum amount of auxiliary information being sent to the verifying users.

The rest of the paper is organized as follows. In Section 2, we present two concepts of related cryptographic primitives and give a concise description on the internal structure of JPEG2000 codestreams. We also show the construction of the Merkle hash tree and how to use it to authenticate sub-sets of data. In Section 3, we first introduce our basic authentication scheme, and then improve it based on user request strategies and patterns. In Section 4, we apply our scheme in the third party publication scenario. In Section 5, we describe briefly the results of our experiments. Section 6 contains our concluding remarks.

2. PRELIMINARIES

2.1 Two Cryptographic Primitives

The following cryptographic primitives will be used in the rest of the paper.

One-way hash function: A hash function takes a variable-length input string and converts it to a fixed-length output string, called a hash value. A one-way hash function, denoted as $h(.)$, is a hash function that works in one direction: it is easy to compute a hash value $h(m)$ from a pre-image m; however, it is hard to find a pre-image that hashes to a particular hash value. There are many existing one-way hash functions, such as MD5[15] and SHA[16]. The hash value of a one-way hash function is also called a message digest. We will use the terms hash, hash value and message digest interchangeably.

Digital signature: A digital signature algorithm is a cryptographic tool for generating non-repudiation evidence, authenticating the integrity of the signed message as well as its origin. In a digital signature algorithm, a signer keeps a private key secret and publishes the corresponding public key. The private key is used only by the signer to generate digital signatures on messages and the public key is used by anyone to verify the signatures on messages. The digital signature algorithms mostly used are RSA [17] and DSA [18].

2.2 Structures of JPEG2000 Codestreams

A JPEG2000 image codestream is organized hierarchically with several kinds of structural elements - tiles, components, tile-components, resolution levels, precincts, layers and packets [1, 3].

Tiles: JPEG2000 allows an image to be divided into smaller rectangular regions known as tiles, each of which is treated as a small independent image for the purpose of compression. Tiles are used to partition the image data into spatial regions. We denote tiles of an image as T_0, ..., T_{n_t-1}, where n_t is the number of tiles of the image.

Components and Tile-components: A component refers to an element of a color space, such as R, G, B. A tile-component is all the samples of a given component in a tile. There is a tile-component for every component and every tile. If a tile is partitioned into n_c tile-components, then we denote these tile-components as C_0, ..., C_{n_c-1}.

Resolution levels: Given a tile-component, a multiple-level dyadic wavelet transform is performed. The first wavelet transform decomposes a tile-component into four sub-bands LL_1, LH_1, HL_1, HH_1 where LL_1 is the lowest frequency sub-band. The second wavelet transform decomposes LL_1 into another four sub-bands LL_2, LH_2, HL_2, HH_2. Applying the wavelet transform continuously on each LL sub-band generates a series of sub-bands belonging to different transform levels. A $(n_r - 1)$-level wavelet transform generates n_r sets of sub-bands, denoted as $R_0 = \{LL_{n_r-1}\}$, $R_1 = \{LH_{n_r-1}, HL_{n_r-1}, HH_{n_r-1}\}, ..., R_{n_r-1} = \{LH_1, HL_1, HH_1\}$. We refer to R_i as resolution level i.

Resolutions: The sub-image constructed from R_0 is a small "thumbnail" of the original image. The sub-image constructed from R_0 and R_1 together is a bigger "thumbnail" of the original image. The sub-image constructed by R_0, R_1 and R_2 is an even bigger "thumbnail" of the original image. Continue like this, we get a series of "thumbnails" and the last "thumbnail" is the original image, which is constructed from R_0, R_1, ..., R_{n_r-1}. We refer to these "thumbnails" as sub-images of resolution 0, resolution 1, ..., and resolution $n_r - 1$, respectively. Note that resolution level and resolution are two different concepts according to our definitions.

Precincts: Each resolution of a tile-component is partitioned into several rectangle regions, called precincts. Precincts are used to make it easier to access the wavelet coefficients corresponding to a particular spatial region of the image. If a resolution level is partitioned into n_p precincts, then these n_p precincts are denoted as P_0, ..., P_{n_p-1}.

Layers and Quality levels: Each precinct goes through a number of coding passes, and a precinct contributes some data for each pass. All the data assembled during a coding pass in a tile forms a layer, which represents an image quality increment. Assuming that a tile is partitioned into n_l layers, we denote these layers as L_i, $i = 0, 1, ..., n_l - 1$. A quality level is composed of a set of continuous layers. Specifically, quality level i, $0 \leq i \leq n_l - 1$, is composed of the set of layers from L_0 to L_i. The sub-image with the quality level 0 has the lowest quality, and the sub-image with the quality level $n_l - 1$ is the original image which has the highest quality.

Packets: Packets are the most fundamental building block of JPEG2000 codestreams. A JPEG2000 codestream can be viewed as a set of packets. For a given tile, four parameters - component, resolution level, precinct and layer, uniquely identify a packet and the packet contains the data corresponding to these four parameters.

Figure 1 shows the packet generation process for a codestream. The process consists of five steps:

1. Decompose a tile into several tile-components.
2. Transform a tile-component into several resolution levels.
3. Partition a resolution level into several precincts.
4. Partition a precinct into several layers.
5. Form a packet from the bit stream corresponding to a given tile-component, resolution-level, precinct and layer.

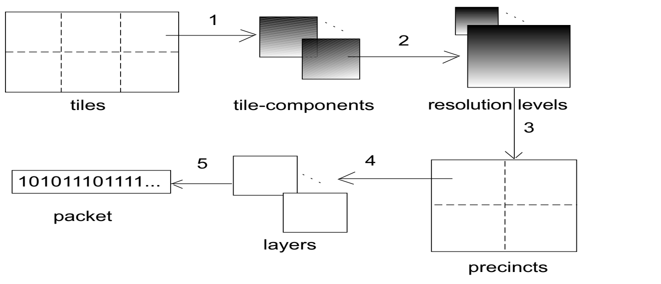

Figure 1: The packet generation process in codestream.

From Figure 1 we see that, a packet in a given tile is uniquely identified by four parameters: C (component), R (resolution level), P (precinct) and L (layer). The packets of an image codestream are sorted with respect to these four parameters in some orders, called progression orders. JPEG2000 allows five progression orders - LRCP, RLCP, RPCL, PCRL and CPRL. For example, a tile with progression order LRCP can be constructed by writing the packets using four nested loops (see Table 1). The innermost loop is precinct, followed by component, followed by resolution level, and finally by layer.

2.3 The Merkle Hash Tree

We illustrate the construction and application of the Merkle hash tree with a simple example. The reader is refereed to [10] for detailed descriptions.

To authenticate data values $n_1, ..., n_w$, the data source constructs the Merkle hash tree as depicted in Figure 2 assuming that $w = 8$. Each node in the tree is assigned a value. The values of the 8 leaf nodes are the message digests, $h(n_1), ..., h(n_8)$, respectively, of the data values under a one-way hash function $h(.)$. The value of each internal node of the tree is derived from its child nodes. For example, the values of node A and node B are $h_a = h(h(n_1)|h(n_2))$ and $h_b = h(h(n_3)|h(n_4))$ respectively, where "|" denotes concatenation. And, the value of node C is $h_c = h(h_a|h_b)$. The data source completes the levels of the tree recursively from the leaf nodes to the root node.

The value of the root node E is $h_e = h(h_c|h_d)$ which is used to commit to the entire tree. It can be used to authenticate any subset of the data values $(n_1, ..., n_8)$, in conjunction with a small amount of auxiliary information. For example, a client, who is assumed to have the authentic root value

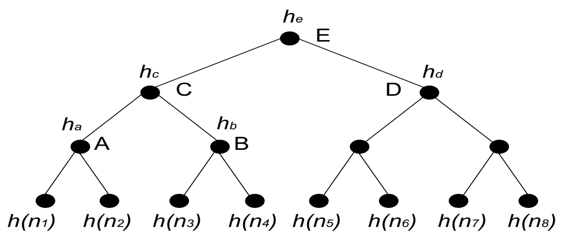

Figure 2: An example Merkle hash tree.

h_e, requests for n_3 and requires the authentication of the received n_3. Besides n_3, the source sends the auxiliary information $h(n_4)$, h_a and h_d to the client. The client can then check the authenticity of the received n_3 as follows. The client first computes $h(n_3)$ and $h(h(h_a|h(h(n_3)|h(n_4)))|h_d)$ and then checks if the latter is the same as the root value h_e. If this check is positive, the client accepts n_3. In general, to authenticate the data value n_i, the auxiliary information is the values of all the sibling nodes of those nodes on the path from the leaf node $h(n_i)$ to the root.

The Merkle hash tree can prevent an adversary, who impersonates as the source, from sending bogus data to the client. In the example of above, an adversary impersonating as the source can not sends a bogus n_3' to the client, because he can not find $h(n_4)'$, h_a' and h_d' such that

$$h(h(h_a'|h(h(n_3')|h(n_4)'))|h_d') = h_e,$$

since $h(.)$ is a one-way hash function. More details about the security of the Merkle hash tree are described in [10]. For the same reason, the integrity of the transferred data can be guaranteed by the Merkle hash tree, i.e, any tamper of the transferred data can be detected.

Though the Merkle hash tree in our example is a binary and balanced tree, generally, it can be non-binary and unbalanced.

3. OUR AUTHENTICATION SCHEME

From the description given in Section 2.3, we note that the root value of the Merkle hash tree must be forwarded to the receiver in an authentic manner; otherwise any Merkle tree based authentication scheme will not be secure [10]. In our authentication scheme, we assume that the data source has a pair of private and public keys in a digital signature scheme and the public key can be distributed to receivers through authentic channels using, for example, a public key infrastructure [19].

The operation of our authentication scheme can be summarized as follows. The source of a JPEG2000 codestream first constructs a Merkle hash tree of the codestream and then computes a signature on the root value of the tree. Upon request of a user, the source sends the packets of the requested sub-image (e. g., the image with a certain resolution and quality layer), the signature on the root value, and a small amount of auxiliary information to the user who can then successfully verify the authenticity of the received sub-image based on the signature and the auxiliary information. Note that the source only needs to compute one signature per JEPG2000 codestream and this single signature allows users to authenticate all the possible sub-images generated from the same codestream. Hence, our authen-

```
for each l = 0, ... , $n_l - 1$                       //$n_l$ is the number of layers
    for each r = 0, ... , $n_r - 1$                   //$n_r$ is the number of resolution levels
        for each c = 0, ... , $n_c - 1$               //$n_c$ is the number of components
            for each p = 0, ... , $n_p - 1$           //$n_p$ is the number of precincts
                packet for component c, resolution r, precinct p, layer l
```

Table 1: The sort of packets under the progression order LRCP.

tication scheme achieves the design objective of "sign once, verify many ways".

In the following, we will omit the description of digital signature generation and verification. Instead, we will focus on the construction of the Merkle hash tree for JPEG2000 codestreams. We first describe how to construct a basic Merkle hash tree for authenticating generic JPEG2000 codestreams. We then introduce two optimizations of the basic Merkle hash tree.

3.1 The Basic Merkle Hash Tree for JPEG2000 codestreams

Figure 1 in Section 2.2 shows clearly a hierarchical structure of JPEG2000 codestreams: an image is partitioned into tiles, tiles into tile-components, tile-components into resolution levels, resolution levels into precincts, finally precincts into packets. Assume that an JPEG2000 image has n_t tiles, each tile has n_c tile-components, each tile-component is decomposed into n_r resolution levels, each resolution level is partitioned into n_p precincts, and each precinct is partitioned into n_l layers, the basic Merkle hash tree for this image codestream is depicted in Figure 3. This tree has 6 levels which correspond, from top to bottom, to the root node, tiles, tile-components, resolution levels, precincts and layers, respectively.

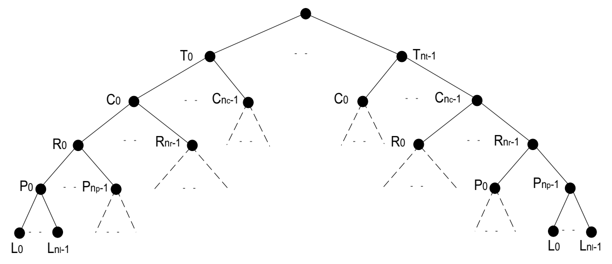

Figure 3: The basic Merkle hash tree for a JPEG2000 codestream.

Recall that, given a tile, four parameters - component, resolution level, precinct and layer, uniquely identify a packet. Therefore, the nodes on the unique path from the root to a leaf node in Figure 3 identify a unique packet. We assign the message digest of the packet under a one-way hash function as the value of the leaf node. For example, the nodes on the path from the root to the leftmost L_0 node in Figure 3 are T_0, C_0, R_0, P_0 and L_0. Hence the value of this leaf node is the message digest of the packet which corresponds to tile 0, tile-component 0, resolution level 0, precinct 0 and layer 0. Once the values of all the leaf nodes are assigned, the values of the other nodes, including that of the root, can be calculated recursively.

A JPEG2000 codestream header specifies important parameters of the codestream such as size, number of layers, number of resolutions and the progression order [1]. It is important to protect the integrity of the header in order to correctly decode the codestream. This can be achieved by hashing the header together with the root value of the Merkle hash tree and let the owner sign the output of the hash function. However, to keep the presentation simple, we will not mention this explicitly in the rest of the paper.

We use an example to further illustrate the above description. Consider an image with 2 tiles, 2 tile-components per tile, 3 resolution levels per tile-component, 2 precincts per resolution level and 3 layers per precinct. Its basic Merkle hash tree is given in Figure 4. We first compute the value for each leaf node. For example, the third leaf node from the left labeled L_2 corresponds to the packet which belongs to tile 0, component 0, resolution level 0, precinct 0 and layer 2, since the nodes on the path from the root to this leaf are T_0, C_0, R_0, P_0 and L_2. Then we can compute its node value as the message digest of the corresponding packet. After computing all the leaf node values, we can compute the non-leaf node values level by level up to the root.

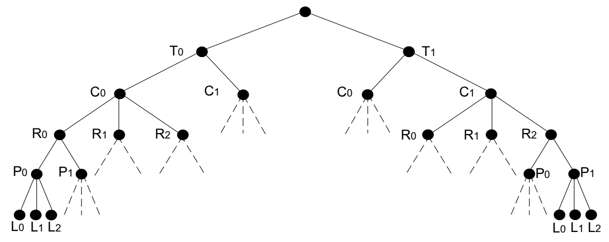

Figure 4: The basic Merkle hash tree of an image with 2 tiles, 2 tile-components per tile, 3 resolution levels per tile-component, 2 precincts per resolution level and 3 layers per precinct.

Now we can authenticate various sub-images using this basic Merkle hash tree. For example, if a user requests the sub-image of resolution 0, i.e. all the packets of resolution level 0, we first send to him the packets belonging to resolution level 0, then send to him auxiliary information that are the node values of those nodes labeled as R_1 and R_2 in Figure 4. Since each C node has 2 such child nodes and there are 4 C nodes, the auxiliary information consists of 8 hash values. As another example, if the user requests a sub-image of quality level 0, i.e. all the packets of layer 0, we first send to him the packets belonging to layer 0, then send to him auxiliary information that are the node values of those nodes labeled as L_1 and L_2 in Figure 4. The auxiliary information this time consists of 48 hash values. In the two following sections, we will modify this basic Merkle hash tree to reduce the amount of auxiliary information.

3.2 Adaption to Resolution-Level-Requests and Layer-Requests

A request for only one of the four parameters - component, resolution level, precinct and layer, is called a single-parameter request. A request for more than one parameters is called a multiple-parameter request. JPEG2000 allows both single- and multiple-parameter requests. In addition, we call a single-parameter request for resolution level as a resolution-level-request, that for layer as a layer-request, that for precinct as a precinct-request, and that for component as a component-request.

In this section, our interest is on resolution-level-requests and layer-requests. Recall the discussion in Section 2.2 that resolution level and resolution are two different concepts: a resolution level only represents incremental data, but a resolution represents a meaningful sub-image. Therefore, almost all resolution-level-requests would ask for the sets of continuous resolution levels starting from resolution level 0. Similarly, almost all layer-requests would ask for the sets of continuous layers starting from layer 0.

Based on the above observation, we modify the basic Merkle hash tree of Figure 3 for resolution-level-requests and layer-requests using the algorithm shown below.

Algorithm 1:

```
1)Move the subtree rooted at $R_i$, $i = 1, ..., n_r - 1$, below
  $R_{i-1}$ such that $R_i$ becomes the right most child of $R_{i-1}$.
  For example, when $i = 1$, move the subtree rooted at
  $R_1$ below $R_0$ and let $R_1$ be the right most child of $R_0$.
2)Move the subtree rooted at $L_i$, $i = 1, ..., n_l - 1$, below
  $L_{i-1}$ such that $L_i$ becomes the right most child of $L_{i-1}$.
  For example, when $i = 2$, move the subtree rooted at
  $L_2$ below $L_1$ and let $L_2$ become the right most child of
  $L_1$.
3)Add a left child node $PK_i$ to each $L_i$, $i = 0, ..., n_l - 1$ .
```

As in the basic hash tree, each leaf node in the adapted hash tree shown in Figure 5 is assigned the message digest of a unique packet. However, there are multiple nodes of the same type on the path from the root to a leaf node. For example, the nodes on the path from the root to the left most PK_2 node are T_0, C_0, R_0, P_0, L_0, L_1 and L_2. There are 3 nodes of type L on this path. Hence, the method of mapping a leaf node to a packet as described in Section 3.1 needs to be modified here. The modification is simple, we just ignore the nodes of the same type except the one closest to the leaf node. In the example above, we ignore L_0 and L_1 so the nodes on the pruned path are T_0, C_0, R_0, P_0 and L_2. Hence the value of the leftmost PK_2 node is the message digest of the packet corresponding to tile 0, tile-component 0, resolution level 0, precinct 0 and layer 2.

Theorem 1. *If a resolution-level-request (layer-request) asks for resolution levels (layers) 0 to $m - 1$ of an image with n ($m < n$) resolution levels (layers), the amount of auxiliary information required for the tree of Figure 5 is less than or equal to that for the tree of Figure 3.*

Proof. In Figure 3, $n - m$ values of nodes $R_m(L_m)$, ..., $R_{n-1}(L_{n-1})$ are needed to re-compute the root node value, because the value of each C node (P node) depends on all the values $R_0(L_0)$, $R_1(L_1)$, ..., $R_{n-1}(L_{n-1})$. In Figure 5, only one value of node $R_m(L_m)$ is needed, because the value of each C node (P node) depends indirectly only on $R_m(L_m)$. Since $n - m \geq 1$, this theorem is proved. □

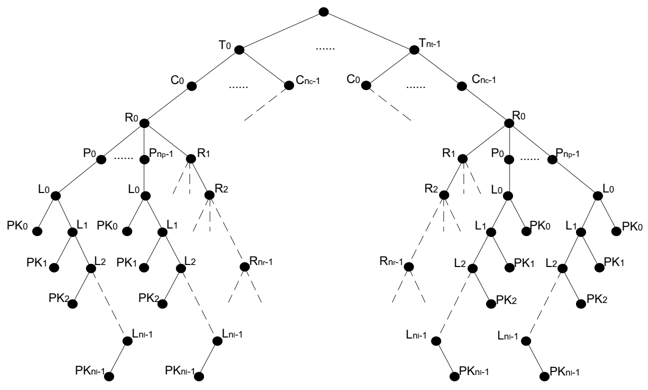

Figure 5: The Merkle hash tree adapted for resolution-level-requests and layer-requests.

Continue with the example of Section 3.1. Applying Algorithm 1, the basic Merkle hash tree in figure 4 is changed to the tree in Figure 6. Now, if a user requests the sub-image of resolution 0, i.e. all the packets of resolution level 0, we first send to him the packets belonging to resolution level 0, then send to him auxiliary information that are the node values of those nodes labeled as R_1 in Figure 6. Since each C node has 1 such child node and there are 4 C nodes, the auxiliary information consists of 4 hash values (while 8 hash values are needed for the tree of Figure 4). If the user requests the sub-image of quality level 0, i.e. all the packets of layer 0, we first send to him the packets belonging to layer 0, then send to him auxiliary information that are the node values of those nodes labeled as L_1 in Figure 6. The auxiliary information consists of 24 hash values (while 48 hash values are needed for the tree of Figure 4).

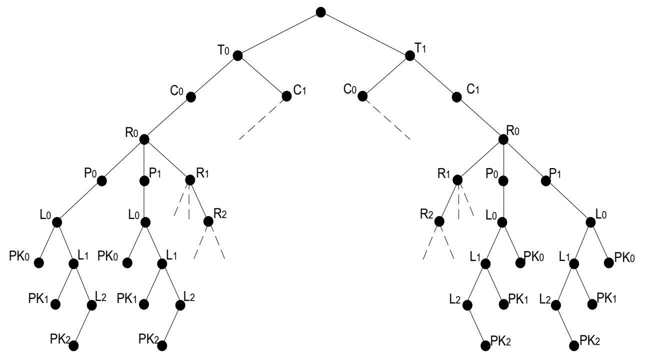

Figure 6: The Merkle hash tree adapted from Figure 4 for resolution-level-requests and layer-requests.

3.3 Adaption to Progression Orders

As has been introduced to in Section 2.2, five types of progression orders are supported by JPEG2000 to suit different application requirements. In an image database browsing application, where progressively refining the quality of an image may be desirable, LRCP is the best progression order because, when the source sends the codestream sequentially, the client constructs a series of sub-images with progressively refined quality. In client-server applications, where different client might demand images at different resolutions, RLCP is the best progression order.

As a result, for image codestreams with different progression orders, the request patterns will be different. For an image codestream with a specific progression order, most requests will be on the first parameter. For an image codestream with progression order LRCP, most requests will be layer-requests such as "request layer 0", "request layer 0 through 1", "request layer 0 through $n_l - 1$"; while for an image codestream with progression order RLCP, most requests will be resolution-level-requests.

Based on this observation, we construct the Merkle hash tree optimized for a given progression order to further reduce the amount of auxiliary information. Denote a progression order as *Ord*, and denote the ith parameter of the progression order as *Ord(i)*, $i = 1, 2, 3, 4$. As an example, if $Ord = LRCP$, then *Ord(1)* $= L$, *Ord(2)* $= R$, *Ord(3)* $= C$ and *Ord(4)* $= P$. The algorithm for optimizing the Merkle hash tree for a given progression order *Ord* is as follows.

Algorithm 2:

```
1) Let the children of the root be $T$ nodes.
2) Let the children of each $T$ node be Ord(1) nodes.
3) Let the children of each Ord(1) node be Ord(2) nodes.
4) Let the children of each Ord(2) node be Ord(3) nodes.
5) Let the children of each Ord(3) node be Ord(4) nodes.
   At this point, we have the basic Merkle hash tree
   arranged by Ord.
6) Move the subtree rooted at $R_i$, $i = 1, ..., n_r - 1$ below
   $R_{i-1}$ such that $R_i$ becomes the right most child of $R_{i-1}$.
7) Move the subtree rooted at $L_i$, $i = 1, ..., n_l - 1$ below
   $L_{i-1}$ such that $L_i$ becomes the right most child of $L_{i-1}$.
8) If Ord(4) $= L$, then add a left child node $PK_i$ to each
   $L_i$ node, $i = 0, ..., n_l - 1$. If Ord(4) $= R$, add a child
   node $PK_i$ to each $R_i$ node, $i = 0, ..., n_r - 1$.
```

The assignment of message digests to leaf nodes follows the same approach of Section 3.2. Applying Algorithm 2 to an image with progression order LRCP, we obtain the hash tree shown in Figure 7.

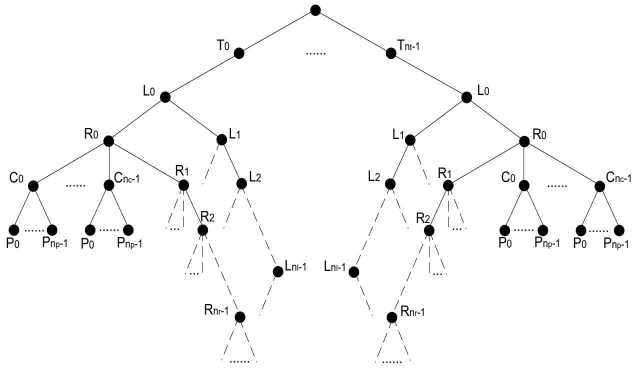

Figure 7: The Merkle hash tree optimized for progression order LRCP.

THEOREM 2. *Let E_1 and E_2 be two node types and $tree_1$ and $tree_2$ be two Merkle hash trees. The tree $tree_1$ has n_1 nodes of type E_1, labeled as $E_{1,1}, ..., E_{1,n}$, and the subtree rooted at each node of type E_1 has n_2 child nodes of type E_2. The tree $tree_2$ has n_2 nodes of type E_2, and the subtree rooted at each node of type E_2 has n_1 child nodes of type E_1, labeled as $E_{1,1}, ..., E_{1,n}$. If the auxiliary information for an E_1-request are the node values labeled as $E_{1,i_1}, ..., E_{1,i_m}$, then the amount of auxiliary information corresponding to the request for $tree_1$ is less than that for $tree_2$.*

PROOF. The amount of auxiliary information for $tree_1$ is m, but that for $tree_2$ is $m \times n_2$. Since $m < m \times n_2$, the theorem is proved. $\square$

Theorem 2 tells us that the most frequently requested node type should be placed as high as possible in the hash tree in order to reduce the amount of auxiliary information. This justifies the hash tree construction process specified by Algorithm 2.

Consider the example of Section 3.1 again. Its hash tree optimized for the progression order LRCP is shown in Figure 8. Let us compare the average amount of auxiliary information for the trees in Figure 6 and Figure 8. The result is shown in Table 2.

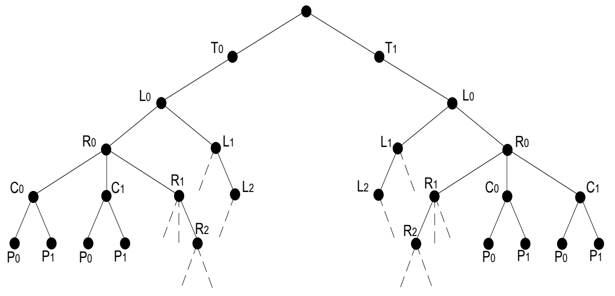

Figure 8: The Merkle hash tree adapted from Figure 4 for progression order LRCP.

First, consider the tree of Figure 8. It is easy to see that a layer-request asking for layer 0 needs 2 node values as auxiliary information, a layer-request asking for both layer 0 and 1 also needs 2 node values as auxiliary information, and a layer-request asking for layer 0, layer 1 and layer 2 needs no auxiliary information. Assuming that the frequencies of these three layer-requests are the same, then the average amount of auxiliary information of layer-requests is 4/3 node values. Similarly, the average amount of auxiliary information of resolution-level-requests is 4 node values, that of component-requests is 9 node values, and that of precinct-requests is 18 node values.

Next, consider the tree of Figure 6. Under the same assumption as above, the average amount of auxiliary information of layer-requests is 16 node values, that of resolution-level-requests is 8/3 node values, that of component-requests is 1 node values, and that of precinct-requests is 6 node values.

The first row of Table 2 contains the average amount of auxiliary information of the tree in Figure 8 under three request frequency distributions respectively. The second row contains that of the tree in Figure 6 under the same three request frequency distributions. Consider the first request frequency distribution as an example, where 90% are layer-requests, 5% are resolution-level-requests, 3% are component-requests and 2% are precinct-requests, the average amount of auxiliary information for the tree of Figure 8 is $90\% \times 4/3 + 5\% \times 4 + 3\% \times 9 + 2\% \times 18 = 2.03$, while that for the tree of Figure 6 is $90\% \times 16 + 5\% \times 8/3 + 3\% \times 1 + 2\% \times 6 = 14.68$. From this table, we see clearly that the tree of Figure 8 requires much less auxiliary information compared to the tree of Figure 6.

4. THE THIRD-PARTY PUBLICATION MODEL FOR JPEG2000 IMAGES

In this section, we use a third-party publication model to illustrate the practical applications of our JPEG2000 image codestreams authentication scheme. The model is shown in Figure 9, where an image owner prepares JPEG2000 codestreams for a publisher (the third party) to disseminate to clients on demand. The responses from the publisher to clients' requests are sub-images. In security-sensitive applications, it is highly desirable or even mandatory for clients to verify the authenticity of a received response, to make sure that the sub-image is indeed originated from the owner as claimed and that the content of the sub-image has not been modified during the transmission.

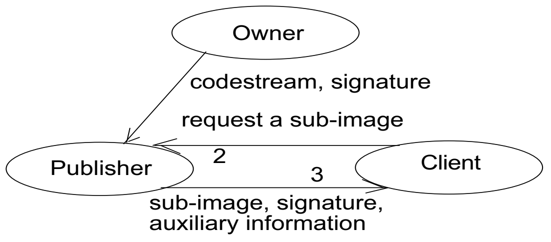

Figure 9: Application of our authentication scheme in the third-party publication model.

A straightforward solution is to let the publisher digitally sign each requested sub-image in real-time. This requires that the publisher be trusted by the clients and does not tamper the original owner's image streams. It also requires that the private signing key be made on-line. Generally speaking, an on-line signing key is vulnerable to both external hacking and insider attacks. Another naive approach is to have the owner pre-compute signatures for all possible sub-images and forward them together with the codestream to the publisher for distribution to clients. This approach is infeasible in practice since there are too many sub-images for a codestream. The approach is also not scalable to large number of codestreams.

Our authentication scheme presented in the last section allows for "sign once, verify many ways"; hence, it addresses exactly the authentication problem in the third party publication model. Using our scheme, the system in Figure 9 operates as follows.

1. The owner of an image codestream constructs a Merkle hash tree using a one-way hash function and generates a digital signature on the root value of the tree. The owner then forwards the codestream and the signature to the publisher.
2. The client requests a sub-image of the codestream.
3. Upon receiving the request, the publisher generates auxiliary information using the same one-way hash function as that of the owner, and sends the requested sub-image, the owner's signature and the auxiliary information to the client.
4. The client verifies the authenticity of the sub-image and if the verification is successful, accepts the sub-image.

Using our authentication scheme in third party publications has three important advantages: 1) the owner only needs to compute the signature once instead of pre-computing signatures for all possible sub-images, 2) the owner need not sign sub-images in real time so that much better security is achieved since the private signing key is not kept on-line and, 3) it requires less trust on the publisher since the publisher is only used for information dissemination, not for generation of digital signatures on behalf of the owner.

Instead of forwarding the entire codestream and its digital signature to the publisher, the owner can also forward a sub-image, the digital signature and the corresponding auxiliary information to the publisher for dissemination to clients. More generally, our model can be extended to the scenario where there exist multiple intermediate publishers: each publisher, except the first and the last, is the "owner" of its succeeding publisher and the "client" of its preceding publisher. As a result, all entities, including owner, publisher and client, form a chain. The owner, the source of the chain, owns the entire codestream. Every other entity except the owner requests a sub-image of the image owned by its preceding entity, which in turn replies with the requested sub-image, the owner's digital signature and the necessary auxiliary information.

5. EXPERIMENT RESULTS

We have designed and implemented a prototype of the above third-party publication model in C++ on a PC with an Intel P4 2.4Ghz processor. The prototype consists of 3 modules - an owner module, a publisher module and a client module. We use MD5 [15] and a 1024-bit RSA [17] as the one-way hash function and the digital signature algorithm, respectively. Both MD5 and RSA can be carried out very efficiently. According to [20], on a PC with a Celeron 850MHz processor under Windows 2000, MD5 achieves 100.7 MBytes/s, and the 1024-bit RSA requires 10.23 milliseconds to generate a signature and 0.32 milliseconds to verify it. Most tested images we use has 1 tile, 3 components, 7 resolution levels for each tile-component, 16 precincts for each resolution level on average, and 10 layers. Their storage is about 200k bytes per image.

The owner module takes a JPEG2000 image codestream as input, generates the Merkle hash tree, signs the root value, and forwards the codestream and the digital signature to the publisher module. Our owner module can generate the Merkle hash trees and signatures for 400 images per second. The owner module embeds the 1024-bit signature into each image delimited by the marker 0xfffe (JPEG2000 uses two byte markers, with the first byte as 0xff, to delimit and signal the characteristics of codestreams). As a result, our authentication scheme only needs 130 bytes storage (the 1024-bit signature plus the 2-byte marker) for the authentication data per image.

The publisher module manages the signed JPEG2000 image codestreams provided by the owner module and is responsible for distributing them to the client module. When the publisher module receives a request from the client module, it analyzes the request, computes auxiliary information and sends to the client the requested sub-image, the signa-

	L-requests(90%)
	R-requests(5%)
	C-requests(3%)
	P-requests(2%)
		L-requests(70%)
		R-requests(20%)
		C-requests(6%)
		P-requests(4%)
			L-requests(50%)
			R-requests(30%)
			C-requests(15%)
			P-requests(5%)

Figure 8	2.03	2.99	3.75
Figure 6	14.68	11.03	9.13

Table 2: The average amount of auxiliary information of the two trees in Figure 8 and Figure 6 under three request frequency distributions.

ture and the auxiliary information. In our experiment, we assumed that 90% of the client requests are single-parameter requests for the first parameter of the progression order and 10% of the client requests as multiple-parameter requests. For single-parameter requests, the auxiliary information consists of 1 hash value for codestreams with progression order LRCP, RLCP or RPCL, 2 hash values for CPRL and 8 hash values for PCRL, respectively, on the average. For multiple-parameter requests, the average auxiliary information is 12 hash values.

After receiving the sub-image, the signature and the auxiliary information, the client module verifies the authenticity of the sub-image by reconstructing the Merkle hash tree and comparing the signed root hash value and the reconstructed root hash value.

Any tamper on an image can be detected by our client model. Figures 10 and 11 show a simple example of tamper detection. The image on the left of Figure 10 is the original image, which has 6 resolution levels and 8 layers, and is signed by the owner's module. The image on the right is the tampered version of the original image (note that the difference is the string "tampered" in the right image). Assume that the client module requests a sub-image with resolution 4 (i.e resolution levels from 0 through 4) and quality level 3 (i.e layers from 0 through 3). The un-tampered sub-image is shown on left in Figure 11. However, because the original image is tampered, the sub-image the client module receives is the one on the right in Figure 11. Our client module detects the tamper successfully and rejects it.

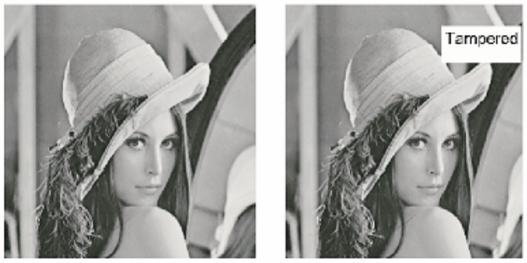

Figure 10: The original image and its tampered version.

6. CONCLUSIONS

Image compression plays a central role in modern multimedia communications and compressed images arguably represent the dominant source of Internet traffic today. JPEG2000 is an advanced image compression standard designed as the successor of JPEG in many of its application areas. In

Figure 11: The original sub-image and its tampered version.

this paper, we have introduced a scheme for authenticating JPEG2000 image codestreams. The scheme is designed based on the Merkle hash tree and digital signatures. Utilizing the hierarchical structure of a JPEG2000 codestream, we constructed in Section 3.1 the basic Merkle hash tree which can be used to authenticate any sub-image from one codestream using just one digital signature and relatively small amount of auxiliary information.

In JPEG2000, the resolution levels and layers in codestream are formed such that images with large resolution or higher quality can be obtained by requesting incremental data. Based on this observation, we presented in Section 3.2 the modified Merkle hash trees which are specially adapted for resolution-level-requests and layer-requests. The adapted trees require smaller amount of auxiliary information compared with the basic Merkle hash tree. By taking advantage of user request patterns, we further modified in Section 3.3 the Merkle hash trees of Section 3.2 to make them more efficient for images with specific progression orders.

To illustrate the practical applications of our authentication scheme, we showed in Section 4 how it can be used to authenticate JPEG2000 images in a third party publication model. A prototype of the third party publication model using our authentication scheme has been implemented and the system demo was given to the plenary session of the JPEG2000 standards meeting held during 10-15th March 2003 in Seoul, South Korea.

Finally we remark that although our scheme is presented in the context of authenticating JPEG2000 codestreams, it can be applied for efficient and flexible authentication of any structured data sets, including other image codestreams such as MPEG4 codestreams.

7. REFERENCES

[1] D. S. Taubman and M. W. Marcellin, *JPEG2000 - Image Compression Fundamentals, Standards and Practice*, Kluwer Academic Publishers, 2001.

[2] M. Rabbani and R. Joshi, "An overview of the JPEG 2000 still image compression standard", *Signal Processing: Image Communication*, Vol. 17, No. 1, pp. 3-48, Elsevier, 2002.

[3] "Information technology - JPEG 2000 image coding system", *ISO/IEC International Standard 15444-1*, ITU Recommendation T.800, 2000

[4] B. Schneier, *Applied Cryptography*, John Wiley & Sons, 1996.

[5] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[6] R. Grosbois, P. Gerbelot and T. Ebrahimi, "Authentication and Access Control in the JPEG 2000 Compressed Domain", *Proc. of the SPIE 46th Annual Meeting, Applications of Digital Image Processing XXIV*, Vol. 4472, pp. 95-104, 2001.

[7] C. Y. Lin and S. F. Chang, "Semi-Fragile Watermarking for Authenticating JPEG Visual Content", *SPIE Security and Watermarking of Multimedia Contents II EI '00*, 2000.

[8] Q. Sun and S. F. Chang, "Semi-fragile image authentication using generic wavelet domain features and ECC", *Proc. of ICIP 2002*, 2002.

[9] Q. Sun, S. F. Chang, M. Kurato and M. Suto, "A quantative semi-fragile JPEG2000 image authentication system", *Proc. of ICIP 2002*, 2002.

[10] R. C. Merkle, "A certified digital signature", *Proc. of Advances in Cryptology-Crypto '89*, Lecture Notes on Computer Science, Vol. 0435, pp. 218-238, Spriner-Verlag, 1989.

[11] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls and G. Stubblebine, "Flexible authentication of XML documents", *Proc. of the 8th ACM conference on Computer and Communication Security*, pp. 136-145, 2001.

[12] M. T. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an Authenticated Dictionary with Skip Lists and Commutative Hashing", *Proc. of DISCEX II'01*, Vol. 2, pp. 1068-1083, 2001.

[13] M. Naor and K. Nissim. "Certificate Revocation and Certificate Update", *Proc. of the 7th USENIX Security Symposium*, pp. 217-230, 1999.

[14] P. Devanbu, M. Gertz, C. Martel and S. Stubblebine, "Authentic Third-party Data Publication", *Proc. of the 14th IFIP WG11.3 Working Conference in Database Security*, IFIP Conference Proceedings, Vol. 201, pp. 101-112, Kluwer, 2001

[15] R. Rivest, "The MD5 Message Digest Algorithm", *RFC 1321*, 1992

[16] National Institure of Standards and Technology, "Secure Hash Standard (SHS)", FIPS Publication 180-1, 1995.

[17] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, Vol. 21, No. 2, pp. 120-126, 1978.

[18] National Institure of Standards and Technology, "Proposed Federal Information Processing Standard for Digital Signature Standard (DSS)", *Federal Register*, Vol. 56, No. 169, pp. 42980-42982, 1991.

[19] R. Housley, W. Ford, W. Polk and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", *RFC 2459*, 1999.

[20] http://www.eskimo.com/~weidai/benchmarks.html.