A SYSTEM FOR END-TO-END AUTHENTICATION OF ADAPTIVE MULTIMEDIA CONTENT

Takashi Suzuki,[1] Zulfikar Ramzan,[2] Hiroshi Fujimoto,[1] Craig Gentry,[2] Takehiro Nakayama,[1] and Ravi Jain[2]

[1]*NTT DoCoMo Media Computing Group*
lastname@mml.yrp.nttdocomo.co.jp

[2]*DoCoMo Communications Laboratories, USA*
lastname@docomolabs-usa.com

Abstract We present a multimedia content delivery system that preserves the end-to-end authenticity of original content while allowing content adaptation by intermediaries. Our system utilizes a novel multi-hop signature scheme using Merkle trees that permits selective element removal and insertion. To permit secure element insertion we introduce the notion of a placeholder. We propose a computationally efficient scheme to instantiate placeholders based on the hash-sign-switch paradigm using trapdoor hash functions. We developed a system prototype in which the proposed signature scheme is implemented as an extension of the W3C XML signature standard and is applied to content meta-data written in XML. Evaluation results show that the proposed scheme improves scalability and response time of protected adaptive content delivery systems by reducing computational overhead for intermediaries to commit to the inserted element by 95% compared to schemes that use conventional digital signatures.

Keywords: Digital signatures, content adaptation, multimedia security

1. Introduction

The popularity of mobile internet services, such as NTT DoCoMo's i-mode [1], has dramatically increased the amount of content delivered to mobile devices. The recent proliferation of third-generation (3G) mobile networks has not only accelerated the increase but also made richer (i.e., bandwidth and CPU intensive) multimedia content available to mobile devices. Due to various service contexts or user preferences that mobile devices can signal to service providers, coupled with the usual mobile device constraints (e.g., viewing time, battery life, and display), content adaptation is expected to play an important role in multimedia content delivery for mobile environments [2]. While such adaptation is useful, there is a cost. In particular, existing systems cannot handle dynamic data adaptation while preserving end-to-end security. The IETF

OPES working group pointed out security issues related to services deployed at application-level network intermediaries [7]. The present paper addresses integrity among the issues. Consider digital signatures - once data is signed, subsequent modifications to it invalidate the signature. *Our goal is to concurrently achieve multimedia content adaptation and end-to-end authenticity.*

We assume a multimedia streaming system where meta-data specifying how media components are handled is provided prior to actual content delivery. Adaptation for such a system can be performed at the meta-data level and the media-data level. For media-data adaption, one may apply transcoding techniques such as multi-rate switching or scalable compression. For meta-data adaptation, one may manipulate the composition of audio and video components in the scene according to user preferences or service contexts.

Several works provide related but different features for adaptive content protection. Teranishi et al. propose an information sharing system with management of content derivation [10]. The primary content provider in their system (called the tier-1 provider here) manages content adaptation by binding usage rules to the meta-data. Their system does not, however, protect content from malicious corruption; instead, they assume that all modifications are performed on trusted hosts. *We, on the other hand, lift this assumption by employing, at our cryptographic core, a novel multi-hop message signature scheme that enables end-to-end authenticity of the usage rule and meta-data.*

Our new signature scheme incorporates techniques from [22, 12] which permit selective adaptive content removal. Merkle trees are used to create a message digest, and deletions involve creating a small cover for the subset of removed data items. Our contribution is to enable secure insertion of secondary content by extending the Merkle hash tree based signature scheme to support placeholders where the tier-2 provider can add its content. We propose a computationally efficient scheme to accommodate the placeholder based on the hash-sign-switch paradigm utilizing trapdoor hash functions [18].

In this paper we focus on the meta-data level signature scheme, although media-data level signatures for streaming media has additional challenges such as delay and scalability – see [4] for a discussion of such issues.

The paper is organized as follows. Section 2 discusses related work. Section 3 explains the proposed system architecture. Section 4 describes the multi-hop signature at our cryptographic core. Section 5 details the system prototype we built and gives performance results. Section 6 makes concluding remarks.

2. Background and Related Work

Many content protection systems [5, 21] take an approach where a content author packages multimedia content with meta-data that contains usage rights information regarding how the content should be used. However, these exist-

ing systems presume content delivery with end-to-end consistency and thwart malicious entities who wish to tamper with content or meta-data; thus they do not permit tier-2 providers to perform adaptation.

The OPES (open pluggable edge services) working group [8] investigates security threats and risks for services deployed at application-level intermediaries, which are relevant to our adaptive content delivery system. In our signature scheme, we introduce placeholders in the Merkle hash tree to support dynamic content insertion by tier-2 providers. Placeholders can be viewed as indicators for OPES callout servers to identify the place for their content. Amongst security threats, this paper addresses policy enforcement at intermediaries that are innately un-trusted by original content providers, and end-to-end authentication of content. Other threats, such as denial of service attacks, end-to-end encryption, are out of the scope of this paper.

To allow adaptation by intermediaries, an information sharing system that enables tier-1 content providers to manage derivative works is proposed in [10]. They define a language for providers to write content usage rules that express the restrictions imposed on derivative works. Modifications are checked against the rules to detect malicious behavior. Their system does not, however, protect content and usage rules from malicious corruption; instead, they assume all modifications are performed on *trusted* hosts, and the content and usage rules never leave a trusted domain. Although one may argue the assumption can be lifted using a conventional signature scheme, the solution assumes that the tier-1 provider trusts tier-2 providers to sign content and usage rules on its behalf. Another solution is that the tier-1 provider sends a signed content with callout indicators that locate content to be inserted. This requires tier-2 providers to register content pointers which they cannot change after the original content is signed.

We alleviate these assumptions by using a novel multi-hop message signature scheme that enables end-to-end authenticity even with content-adapting tier-2 providers. Such trust alleviation enables various service scenarios where tier-2 providers are not trusted entities. For example, it enables personal users to legitimately use commercial music clips to create personal video clips, and distribute them subject to usage policies of original content.

3. Proposed System Architecture

Figure 1 illustrates the basic architecture of our adaptive multimedia content delivery system. It supports both meta-data and media-data content adaptation. The tier-1 provider (T_1) creates media files which constitute multimedia content and generates meta-data containing information on how to access the media files and how to compose them. Meta-data is delivered to user devices through tier-2 providers (T_2). The actual media files are stored in media servers

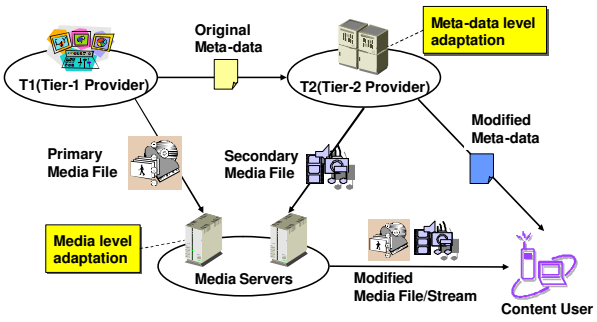

Figure 1. Basic architecture of adaptive content delivery system which supports two-level adaptation: meta-data level and media-data level.

and are delivered to user devices by downloading or streaming. These media servers may be operated by T_1, T_2, or other third-party providers. T_2 may perform adaption on meta-data elements. Each element represents an actual media file; T_2 inserts/deletes a media file to/from a composition by manipulating meta-data without touching the internal content of the files. Examples of such adaptation are dynamic insertion of (targeted) advertisements or removing elements to create a digest.

We assume weak trust between T_1 and T_2. T_2 acquires from T_1 the right to create and re-distribute content derived from the primary content, subject to usage rules specified by T_1. We envision that their relationship is as loose as that between an e-commerce site and consumers. T_1 may have no reason to trust T_2, and presumes it might try to perform illegal modifications on the content and usage rules. Further, T_2 does not trust other tier-2 providers or end-users, and assumes they may try to "remove" T_2's adaptation (e.g., delete its advertisements). We do, however, assume that a trusted end-user media player; e.g., it uses attestation to a trusted platform or a tamper-resistance mechanism (with the usual caveat that such measures can often be circumvented by highly determined adversaries). If the media player detects usage rule violations it takes the appropriate action; e.g., prohibits the client from downloading necessary media files. To realize secure adaptive content delivery under these assumptions, we need a signature scheme with the following properties:

1 T_2 can delete or insert elements from/to meta-data subject to the usage rule specified by T_1. Rule violation must be detected by the verifier.
2 T_2 can insert additional elements only at positions specified by T_1. The verifier can detect insertions to an un-designated position.
3 Once T_2 commits to the element it inserts, the element cannot be altered or deleted without detection by verifiers. (As in any signature scheme, what happens after malicious behavior is detected is orthogonal.)

The “commitment" above does not bind T_2 to a particular element; instead it forms a secure placeholder into which content can be dynamically inserted.

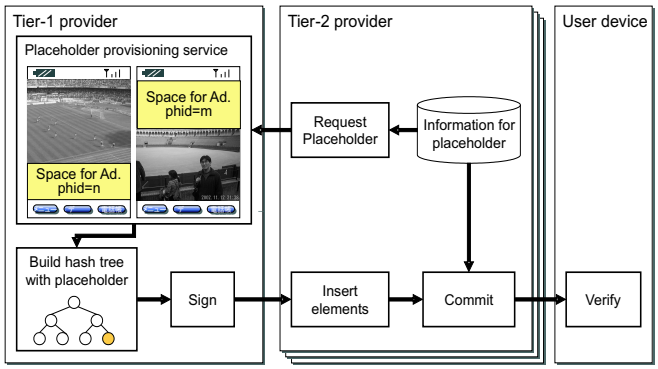

Figure 2. Placeholder request procedure between tier-1 & tier-2 providers; the tier-2 provider sends a request with placeholder information to purchase space.

The next section describes a scheme that achieves these properties. The scheme uses Merkle trees and achieves the first property by binding a hash of the usage rules to the Merkle root prior to signing. The verifier checks both rule compliancy of the modification and signature validation. To realize irreversible inserts, the scheme introduces an extension, called a placeholder, which specifies a position in the meta-data into which a designated entity can insert an additional meta-data element. The placeholder contains information that uniquely identifies the entity who will insert data. After the placeholder is set to the meta-data, T_1 constructs its hash tree and signs the root. Before it signs the meta-data, T_1 must obtain T_2's information for inclusion in the placeholder. This can happen via a directory service or by using a placeholder request procedure between T_1 and T_2. We adopt the second approach, imagining a scenario where T_2 purchases advertisement space by sending a placeholder request message (see Figure 2). Upon receipt of the signed meta-data, T_2 securely inserts an element (e.g., advertisement) to the assigned placeholder.

Although our focus is meta-data adaptation and protection, our system can incorporate the authenticated media-level adaption techniques proposed by Gentry et al. [4] which permit transcoding of the media content itself while preserving end-to-end authenticity.

4. Proposed Signature Scheme

We propose signature schemes that allow one or more T_2s to modify original content by dynamically deleting or inserting elements (subject to T_1's policy), while preserving a receiver's signature verification ability. We first describe Merkle trees and how they permit dynamic deletion. We then introduce placeholders, which allow for dynamic insertion; our main scheme instantiates placeholders using trapdoor hash functions at select Merkle tree leaves. Finally, we discuss two mechanisms that make content insertion irreversible.

Model and Notation. Let $\mathcal{S}$ denote the sender or T_1 who creates and signs the original data and let $\mathcal{R}$ denote the receiver who verifies the signature. The data may pass through a proxy $\mathcal{P}$ such as T_2. Our schemes extend to multiple proxies, but for simplicity, we only consider the case of one $\mathcal{P}$. $\mathcal{P}$ may insert content or remove portions of the data. It further determines what information, if any, $\mathcal{R}$ requires to verify the signature. We implicitly assume $\mathcal{S} \neq \mathcal{P}$, but our schemes work when they are the same. We assume the existence of an open or closed public-key infrastructure where $\mathcal{S}$ has key pair $(\mathsf{Pk}, \mathsf{Sk})$. Here Sk is $\mathcal{S}$'s private key for computing a traditional signature on a message, and Pk is the public verification key. $\mathsf{Sign}(\mathsf{Sk}, M)$ denotes an algorithm that outputs a signature σ on message M under signing key Sk and $\mathsf{Vf}(\mathsf{Pk}, M, \sigma)$ denotes the verification algorithm. $\mathcal{P}$ need not know Pk or Sk. Our schemes may also work with a message authentication code or MAC, in which case both $\mathcal{S}$ and $\mathcal{R}$ share knowledge of a symmetric key (which $\mathcal{P}$ need not know).

Let $\{0,1\}^*$ denote the set of all bit strings. Let M denote the initial content that can be broken up into n blocks which may have different lengths: $M = M_1 M_2 \cdots M_n$, $M_i \in \{0,1\}^*$, $1 \leq i \leq n$. Where convenient, we assume n is a power of 2. In our scheme, the intermediary may choose to either keep or remove an *entire* block, but he cannot perform transformations involving a portion of a block. Let $\mathcal{H}$ denote a *cryptographic hash function* that takes as input a string in $\{0,1\}^*$ and a (fixed and publicly known) v-bit initialization vector (IV), and produces a v-bit output. We assume these cryptographic hash functions are *collision resistant*; i.e., finding two inputs $m_1 \neq m_2$ such that $\mathcal{H}(\mathsf{IV}, m_1) = \mathcal{H}(\mathsf{IV}, m_2)$ is difficult. A practical example of such a cryptographic hash function is SHA-1 [15] which has a 20-byte output and IV.

Merkle Trees, Signing, and Deletion. The Merkle tree associated with M is a balanced binary tree in which each node v is assigned a value $\mathcal{V}(v)$ – we often refer to v and $\mathcal{V}(v)$ interchangeably. There are n leaves, and for each leaf ℓ_i, $\mathcal{V}(\ell_i) = \mathcal{H}(\mathsf{IV}, M_i)$, $1 \leq i \leq n$. For an interior vertex v, $\mathcal{V}(v) = \mathcal{H}(\mathsf{IV}, \mathcal{V}(\mathsf{C}_0(v)) \circ \mathcal{V}(\mathsf{C}_1(v)))$, where $\mathsf{C}_0(v)$ and $\mathsf{C}_1(v)$ are v's left and right children respectively, and $\circ$ denotes concatenation. To sign M, the content creator computes the root value r of the Merkle tree associated with M. The signature is $\sigma = \mathsf{Sign}(\mathsf{Sk}, r)$. Deletion is supported by supplying a modicum of extra verification data so that the verifier can still compute the root of the Merkle tree, as we now describe. First, let M' denote the transformed data after the removal of blocks. The intermediary does the following:

1 Let $S = \{(\ell \mid \ell \text{ is a leaf of a block to dropped.}\}$
2 If there exist $u, v \in S$ such that u, v are siblings in the tree, then set $S = S - \{u, v\} \cup \{w\}$, where w is the parent of u, v. Repeat this until S has no siblings. Suppose that at the end $S = \{w_i \mid 1 \leq i \leq \rho\}$.
3 Let $\mu_i = \mathcal{V}(w_i)$ for $1 \leq i \leq \rho$. $\mathcal{P}$ transmits M', σ, μ_i, and the tree node position for each w_i, $1 \leq i \leq \rho$.

Verification. $\mathcal{R}$ verifies the signature as follows:

1 For each received message block M_{i_k}, compute $y_k = \mathcal{H}(\mathsf{IV}, M_{i_k})$.

2 Consider the set of all hash values computed in the previous step as well the hash values $\mu_1, \ldots, \mu_\rho$. If any pair correspond to siblings, replace the pair with their hash (which corresponds to their Merkle tree parent). Repeat this step until only one value remains – call it r'.

3 Run $\mathsf{Vf}(\mathsf{Pk}, r', \sigma)$.

We show that $r' = r$, from which it is easy to see why the above algorithm works. If one has all the initial blocks, then the above procedure is the standard algorithm for computing the Merkle tree root. Now, observe that whenever $\mathcal{R}$ receives some hashes $\mu_1, \ldots, \mu_\rho$, these come from $\mathcal{P}$ running the *same algorithm* on the subset of missing frames. Therefore, $\mathcal{P}$ and $\mathcal{R}$ have together run the algorithm on all n blocks which yields the Merkle root value.

Insertion via Placeholders. We propose the CS and HSS schemes for realizing placeholders. The former uses conventional public-key signatures (e.g., RSA) and the latter uses the hash-sign-switch [13] technique. The CS scheme is fairly trivial. $\mathcal{S}$ places $\mathcal{P}$'s public key (or instructions on where to retrieve it) in the placeholder block. $\mathcal{S}$ then creates a Merkle-tree digest and signs as described above. $\mathcal{P}$, in turn, attaches its content and signs it separately. $\mathcal{R}$ checks the validity of both signatures. This approach is less efficient than the HSS scheme which we now proceed to describe.

HSS Scheme. Trapdoor hash functions $H_Y(m, r)$ consist of a public key Y and a trapdoor X; they take two arguments and have following properties:

- If X is unknown, there is no efficient algorithm that finds pairs (m_1, r_1) and (m_2, r_2) such that $H_Y(m_1, r_1) = H_Y(m_2, r_2)$, but $m_1 \neq m_2$, except with negligible probability.
- If X is known, there is an efficient algorithm that given m_1, m_2, r_1 with $m_1 \neq m_2$, finds r_2 such that $H_Y(m_1, r_1) = H_Y(m_2, r_2)$.

One can construct such trapdoor hash functions based on the discrete logarithm assumption (DLA) as follows (see [18] for details). Let p, q be primes such that $q|p-1$, and let g be an element of order q in $\mathbb{Z}_p^*$ – parameters are global. The trapdoor is a value x chosen (randomly) from $\mathbb{Z}_q^*$. The public key is $y = g^x \bmod p$. Now, we define $H_y(m, r) = g^m y^r$, which can be computed by anyone. However, for any given $m_1, m_2, r_1 \in \mathbb{Z}_q$, knowledge of the trapdoor is required to efficiently compute an $r_2 \in \mathbb{Z}$ such that $H_y(m_1, r_1) = H_y(m_2, r_2)$ by setting $r_2 = (m_1 - m_2)x^{-1} + r_1$. To create a placeholder, $\mathcal{P}$ sets up a trapdoor hash function and hashes random values, m', r': $TH = H_Y(m', r')$. It sends TH and Y to $\mathcal{S}$. $\mathcal{S}$ treats the received parameters as a message block, and signs everything using the above hash tree technique. To insert content m, $\mathcal{P}$ uses x to compute r such that $H_Y(m, r) = TH$. These values together

with the original signature are sent to $\mathcal{R}$. In turn, $\mathcal{R}$ verifies the original signature using TH as a placeholder, and then determines placeholder validity by checking if $H_Y(m, r) = TH$.

Comparison. HSS is more efficient than CS for $\mathcal{P}$. Assuming the DLA implementation, HSS requires one modular multiplication. CS using 1024-bit RSA keys requires a full-length exponentiation, which involves about 1500 modular multiplications on average. Also, in HSS, $\mathcal{P}$ need not rebuild the hash tree after adding data, but merely attaches the commitment value r to the original signature. $\mathcal{R}$ need only rebuild the hash tree to verify the original signature and hash. In CS, however, $\mathcal{P}$ must re-build the hash tree since malicious entities can replace the added data with any data previously signed by $\mathcal{P}$. $\mathcal{R}$ must rebuild two hash trees from the meta-data with and without the added data. A drawback of the DLA based HSS is that placeholders can only be used once, otherwise x leaks by solving simultaneous equations. A simple modification enables us to use placeholders k times. We generate k public keys $\{y_i = g^{x_i}(\mathrm{mod}p) : 1 \leq i \leq k\}$ and compute hash value $TH = g^{m'} y_1^{r'}(\mathrm{mod}p)$. To use the i^{th} message m_i, $\mathcal{P}$ computes $r_i = (m' + r'x_1 - m_i)x_i^{-1} \bmod q$. $\mathcal{R}$ checks that $g^{m_i} y_i^{r_i} = TH$. The CS placeholder has unlimited reuse.

Preventing Removal. We should also prevent malicious deletion of T_2's inserted content by those who, say, do not want to see advertisements. There are two approaches – each of which is compatible with both HSS and CS. In the first approach, $\mathcal{P}$ signs each of its placeholders regardless of whether it wishes to insert content into the corresponding slot. Then, any placeholder without a corresponding signature constitutes evidence that $\mathcal{P}$'s content was illegitimately deleted. Since only $\mathcal{P}$ can produce signatures corresponding to its own public key (which is embedded in the Merkle tree generated by $\mathcal{S}$), no other parties can remove or modify $\mathcal{P}$'s inserted content without detection.

The second approach, which we have not yet implemented, uses aggregate signatures [19], which is a single signature that convinces any verifier that signer S_i signed message M_i, $1 \leq i \leq n$, for distinct signers and messages. One advantage of aggregate signatures is compactness; ideally, the size of the aggregate signature does not grow at all as n increases. Here, we use a different property of certain aggregate signatures: when two entities (e.g., T_1 and T_2) aggregate their respective signatures, it is impossible for a third party (e.g., a second tier-2 provider or a receiver) to separate or "disaggregate" the signatures. Using this property, T_2 can ensure that its insertion cannot be removed without detection by aggregating its signature on its insertion with T_1's signature on its content. Then, any deletion of T_2's content will be detected by a receiver that attempts to verify the authenticity of T_1's content.

As an example, we consider the BGLS aggregate signature scheme [19], which uses a function $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ called a "pairing", that maps two

elements of an elliptic curve group (or abelian varieties group) $\mathbb{G}_1$ to a second group $\mathbb{G}_2$. (See [19] for details.) We assume that all users of the aggregate signature scheme share certain parameters, such as a point P of order q on the elliptic curve, a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_1$, and a public key s_1P. Similarly, $\mathsf{T_2}$ has key pair (s_2, s_2P). Let M_1 denote $\mathsf{T_1}$'s content (including, of course, the placeholder for $\mathsf{T_2}$), and let M_2 denote $\mathsf{T_2}$'s content. $\mathsf{T_1}$ computes its signature on M_1 as $s_1P_{M_1}$, where $P_{M_1} = H(s_1P, M_1)$; the tier-2 provider similarly computes its signature on M_2 as $s_2P_{M_2}$, where $P_{M_2} = H(s_2P, M_2)$. The aggregate signature is $S_{agg} = s_1P_{M_1} + s_2P_{M_2}$, and it is verified by confirming that $e(S_{agg}, P) = e(P_{M_1}, s_1P)e(P_{M_2}, s_2P)$. It is impossible (even information theoretically) to recover $s_1P_{M_1}$ or $s_2P_{M_2}$ from S_{agg}.

To prevent the removal of $\mathsf{T_2}$'s inserted content, $\mathsf{T_1}$'s original signature (i.e., $s_1P_{M_1}$) must be sent to $\mathsf{T_2}$ over a secure channel; otherwise, anyone can replace the aggregate signature with $\mathsf{T_1}$'s non-aggregated signature. When combining this scheme with HSS, $\mathsf{T_1}$ generates random values for m' and r', and computes $TH = H_Y(m', r')$ as usual. It then generates and stores the tentative signature s_2P_{TH}, where $P_{TH} = H(s_2P, TH)$. To sign M_2, $\mathsf{T_1}$ transmits the r corresponding to M_2 and aggregates s_2P_{TH} with $\mathsf{T_2}$'s signature. If $\mathsf{T_2}$ does not wish to insert content at a placeholder, it need not sign.

Under this approach $\mathcal{P}$'s signature is shorter. The user's verification time is proportional to the number of used placeholders as opposed to the total number allocated. However, verification time may be greater since computing a pairing takes, as a rule-of-thumb, about the same time as five full-length 1024-bit modular exponentiations.

Security Analysis. Our formal security analysis requires three standard assumptions: the scheme used to sign the Merkle root resists existential forgeries under adaptive chosen message attack in the sense of [20]; $\mathcal{H}$ is collision-resistant; and the trapdoor hash function is collision resistant (if the trapdoor is unknown). The last assumption can, in turn, be based on the discrete logarithm assumption. We can theoretically base $\mathcal{H}$ on this assumption as well, though in practice we use SHA-1. The underlying reductions are tight in a concrete security sense. The proof is straightforward and combines techniques from [22] (to address secure removal) and [13] (to address secure insertion).

5. Implementation and Evaluation Results

We built a prototype of our adaptive content delivery system with the proposed signature schemes in Java. We used SMIL [11] for meta-data specifying the composition of media files, and XACML [16] to write usage policies restricting adaptation to the original SMIL file. We adopted XML digital signatures as a basis of our signature scheme for meta-data, and implemented an extension to support Merkle hash trees with placeholders.

Signatures. Figure 3 shows a (simplified) example SMIL document before and after signing. The document includes a placeholder for video with an identifier (`phid`) of 1. The Signature element is written in XML-DSIG with two extensions: a new hash algorithm identifier, `HashTreeConstruction` (for the Merkle tree) and a new element, `TrapdoorHashMethod` (for trapdoor hash function parameters). The policy element specifies allow/deny rules of add and delete operations for each element under the SMIL element. The add operation rule specifies the placeholder identifier and attributes with which the inserted data must comply. The Policy element is bound with the SMIL elements by including its hash value in the Signature element. T_2 sends a SOAP placeholder request message containing the identifer and parameters associated with the placeholder. T_1 copies the parameters from the message to the `PublicValue` element and the `TrapdoorHashValue` before calculating the signature.

```
<?xml version="1.0"?>
<smil>
  <head/>
  <body>
    <seq>
      <par>
        <video src="rtsp://tyer-1/video1.rm"/>
        <video src="rtsp://tyer-1/music1.rm"/>
      </par>
      <par>
        <video phid="1"/>
      </par>
    </seq>
  </body>
</smil>
```

(a) Before signed

```
<?xml version="1.0"?>
<DocumentRoot>
 <Policy/>
 <smil/>
 <Signature>
   <SignedInfo>
     <CanonicalizationMethod/>
     <SignatureMethod/>
     <Reference URI=/DocumentRoot/Policy />
     <Reference URI=/DocumentRoot/smil/head />
     <Reference URI=/DocumentRoot/smil/body>
       <DigestMethod Algorithm="HashTreeConstruction"/>
       <DigestValue> root_node_of_hash_tree </DigestValue>
     </Reference>
     <TrapdoorHashMethod Algorithm="Discrete Log" phid="1">
       <PublicValue> public_values_of_trapdoor_hash </PublicValue>
       <TrapdoorHashValue> trapdoor_hash_value
                          </TrapdoorHashValue>
     </TrapdoorHashMethod>
   </SignedInfo>
   <SignatureValue> Signature </SignatureValue>
 </Signature>
</DocumentRoot>
```

(b) After signed (***Bold: extended part***)

```
<smil>
  <head/>
  <body> <seq>
      <par>
        <video src="rtsp://tyer-1/video1.rm"/>
        <video adaptation="delete"/>
      </par>
      <par> <video phid="1" src="rtsp://tyer-2/xxx.rm"  adaptation="add"/>
      </par>
  </seq> </body>
</smil>
```

(c) After transformed

```
<Signature>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    <Reference URI=/DocumentRoot/Policy />
    <Reference URI=/DocumentRoot/smil/head />
    <Reference URI=/DocumentRoot/smil/body />
    <TrapdoorHashMethod Algorithm=
                    "Discrete Log" phid="1"/>
  </SignedInfo>
  <SignatureValue/>
  <AdditiveSignature phid="1">
    <CommitmentValue>
      Commitment_Value_of_TrapdoorHash
    </CommitmentValue>
  </AdditiveSignature>
</Signature>
```

(d) After commitment

Figure 3. SMIL documents (a) before, (b) after signing, (c) after transformation, and (d) after committment.

Adaptation. T_2 modifies the signed SMIL document subject to the Policy element restrictions. The transformed meta-data is checked against the Policy Element restrictions. If the result is "allow", the added element's commit-

ment value is calculated and set to the element `ArgValue` under the element `AdditiveSignature` element. Otherwise, we interrupt the process.

Verification. The user device verification module is implemented as an HTTP proxy which evaluates a signed document and outputs a SMIL document that can be handled by existing SMIL players such as RealOne [6]. There are three steps in the verification process: signature validation, policy compliance check (identical to what is in the adaptation module), and transformation. The signature validation step reconstructs the hash tree from the leaves to the root. If an element contains an adaptation attribute of "add", it fetches the hash value from the corresponding `TrapdoorHashMethod` element with the same `phid`. After hash tree reconstruction, the value in the `SignatureValue` element is validated. Commitment value for the added element is validated separately. The trapdoor hash value is computed using the added data and the commitment value, and compared with the trapdoor hash value in the Signature element. If the signature is valid, the adaptation by T_2 is checked against the restrictions in the Policy element. The policy compliancy check step in the adaptation process is reused here. If the check succeeds, the signed document is transformed to a standard SMIL format by deleting system-specific elements and attributes. If any of the above steps fails, the verification module sends an error message to the SMIL player and terminates.

Performance. Modules in T_2 are implemented on a 3.06 GHz Pentium 4 machines with 1 GB memory running Redhat Linux 2.4.20. The user device modules are implemented on an 866MHz Pentium III machine with 512 MB memory running Windows XP. Our experiments used 1024-bit DSA-SHA1 in XML-DSIG for both the Hash-Sign-Switch (HSS) and Conventional Signature (CS) schemes. For HSS, the trapdoor hash function also uses a 1024-bit modulus. All results are computed by averaging 10 trials.

Figure 4 shows processing delay in msec of each step in T_2 (left) and the user devices (right). In the figure, "XML-DSIG", "XML-DSIG (hash tree)", "One delete", "One add (Conv.)", and "One add (Trapdoor)" mean processing delay to handle a SMIL document with XML-DSIG, XML-DSIG with hash tree extension but without adaptation, one delete operation, one add operation using CS, and one add operation using HSS, respectively. The SMIL document included 5 leaf elements. In T_2, commitment using CS was implemented using the existing XML-DSIG implementation and took about 439msec for one added element. On the other hand, commitment using HSS took only about 23 msec for the same added element. This commitment step includes insertion of the commitment value to the Signature element. The processing delay of adaptation and the policy compliance check took about 11msec and 10msec respectively, rather insignificant compared to commitments implemented using CS. In the user device, signature verification using the existing XML-DSIG

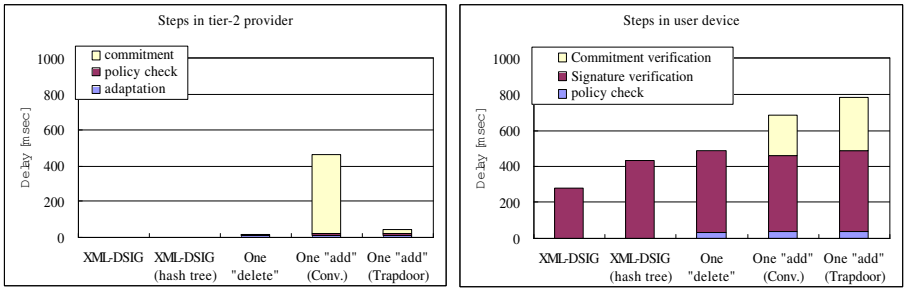

Figure 4. Processing delay in msec of each step in T_2 (left) and the user devices (right).

implementation [9] took 282 msec in our setting. In case of using hash tree construction, the verification delay increased by 52%. The processing delay of the policy check was 38 msec, three times longer than that in T_2 because of lower CPU power. Verification of the commitment using HSS required about 32% more overhead than the commitment using CS.

We see that HSS reduces computational overhead for T_2 to commit to the element added to the original meta-data by 95% (400 msec in our setting). Although this improvement comes at the cost of increased verification overhead on the user device by 32% (70 msec), total end-to-end overhead is reduced by 30% (315 msec). This indicates that the HSS scheme improves scalability and response time of secure adaptive content delivery systems.

6. Conclusions

We presented a protection system for adaptive multimedia content delivery that preserves end-to-end authenticity while allowing content adaptation by intermediaries. We proposed a new multi-hop digital signature scheme, and used it to protect content meta-data and usage rules from illegal modifications. The proposed signature scheme uses Merkle hash trees to allow selective element removal, and achieves secure element insertion by adding a placeholder extension. We also suggested a computationally efficient scheme to instantiate the placeholder based on the hash-sign-switch paradigm using trapdoor hash functions. The proposed scheme can alleviate the trust level from T_1 to tier-2 providers; otherwise T_1 must have complete trust in them not to perform illegal modifications. We envision that this trust alleviation will give flexibility to secure adaptive content delivery services. The evaluation results using our prototype showed that HSS can reduce signature-related overhead in the tier-2 provider (commitment) and end-to-end (signature, commitment, and verification) by 95% and 30% respectively, compared to CS. The proposed signature scheme, thus, contributes to improvement of scalability and response time of adaptive content delivery systems with the content protection scheme.

References

[1] NTT DoCoMo i-mode. http://www.nttdocomo.com/corebiz/imode.

[2] M. Etoh and S. Sekiguchi. MPEG-7 enabled digest video streaming over 3G mobile network. *12th International Packet Video Workshop (PV2002)*, Apr '02.

[3] O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, vol. 33, no. 4, 1986, pp 210-217.

[4] C. Gentry, A. Hevia, R. Jain, T. Kawahara, and Z. Ramzan. End-to-End Security in the Presence of Intelligent Data Adapting Proxies: the Case of Authenticating Transcoded Streaming Media. *To Appear in J. Selected Areas of Communication,* Q1, 2005.

[5] Microsoft Windows Media 9 Series. http://www.microsoft.com/windows.

[6] Real Networks. RealOne player. http://www.realnetworks.com.

[7] IETF RFC 3238. http://www.ietf.org/rfc/rfc3238.txt.

[8] IETF Open Pluggable Edge Services (OPES) Working Group. http://www.ietf.org/html.charters/opes-charter.html.

[9] IBM alphaWorks XML Security Suite. http://www.alphaworks.ibm.com/tech/.

[10] T. Yuuichi, T. Kaori, O. Takeshi, S. Shinji, and M. Hideo. ASIA: Information Sharing System with Derived Content Restriction Management. *IEICE Transactions on Communications (Japanese Edition)*, vol. 428, pp 1463–1475, Aug '03.

[11] W3C Recommendation. Synchronized Multimedia Integration Language (SMIL 2.0). http://www.w3.org/TR/smil20. Aug '01.

[12] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. *CT-RSA, Lecture Notes in Computer Science*, vol. 2271, pp 244-262, 2002.

[13] A. Shamir and Y. Tauman. Improved Online/Offline Signature Schemes. *Proc. of Crypto 2001*, pp 355-367.

[14] W3C Recommendation. XML-Signature Syntax and Processing. http://www.w3.org/TR/xmldsig-core. Feb '02.

[15] National Institute of Standards and Technology, U.S. Department of Commerce. Secure Hash Standard. *Federal Information Processing Standards Publication 180-1,* Apr. 1995.

[16] OASIS Committee. eXtensible Access Control Markup Language v1.0. http://www.oasis-open.org. Feb '03.

[17] R. Merkle. Protocols for Public Key Cryptosystems. *Proc. of the IEEE Symposium on Security and Privacy*, pp 122-134, 1980.

[18] H. Krawczyk and T. Rabin. Chameleon Hashing and Signature. *Proc. of NDSS '2000*.

[19] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. *Proc. of Eurocrypt '03*. LNCS 2656, pp. 416-432.

[20] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing.* 17(2), pp281-308, 1988.

[21] OMA. DRM2.0 Enabler Release. http://www.openmobilealliance.org. Feb '04.

[22] R. Steinfeld, L. Bull and Y. Zheng. Content Extraction Signatures. *Proc. of ICISC 2001*. LNCS, vol.2288, pp.285-304.

[23] W3C Recommendation. SOAP v1.2. http://www.w3.org/TR/SOAP. June '03.

[24] W3C Recommendation. XSL Transformations v1.0. http://www.w3.org/TR/xslt. Nov '99.