Int J Digit Libr (2004) 4: 208–222 / Digital Object Identifier (DOI) 10.1007/s00799-004-0082-z

A Hierarchical Extraction Policy for content extraction signatures

Selectively handling verifiable digital content

Laurence Bull[1,*], **David McG. Squire**[1], **Yuliang Zheng**[2]

[1] School of Computer Science and Software Engineering, Monash University, 900 Dandenong Road, Caulfield, Victoria, 3145, Australia
e-mail: {l.bull,David.Squire}@csse.monash.edu.au
[2] Department of Software and Information Systems, University of North Carolina at Charlotte, Charlotte, NC 28223, USA
e-mail: yzheng@uncc.edu

Published online: 20 October 2004 –

Abstract. Content extraction signatures (CES) enable the selective disclosure of verifiable content from signed documents. We have previously demonstrated a CES Extraction Policy for fragment grouping to allow the document signer to designate which subsets of the original document are valid subdocuments. Extending this ability, we introduce a new *Hierarchical Grouping Extraction Policy* that is more powerful, and for which the encoding is dramatically smaller, than the existing Grouping Extraction Policy. This new Extraction Policy maps naturally onto the hierarchically structured documents commonly found in digital libraries. After giving a motivating example involving digital libraries we then conjecture as to how to enrich their functionality through the use of CESs. We also show how to implement the new extraction policy using XML signatures with a custom transform along with an improved design for the XML signature structure in order to achieve CES functionality.

Keywords: Content extraction signatures – XML Signature custom transforms – Selective content disclosure – Hierarchical Extraction Policy – Privacy-enhancing signatures

1 Introduction

As the Internet burgeons and electronic society emerges, the volume of digital information increases. To cope with the growing flood of data, we need new ways of handling and processing information that are not just electronic analogs of what has been done in the paper-based world.

Documents are merely containers. In the paper-based world, however, the tight binding of the medium and the message makes this distinction hard to see: we tend to think, for example, of a certificate *being* a piece of paper, rather than the facts printed on it. Traditionally, and in most computerized implementations to date, this view has been perpetuated: documents have been viewed and handled as coherent collections of semantically grouped information. Some documents, however, are merely containers of facts, such as a contract, an academic transcript, a non-fiction book or an encyclopedia. It is with the verifiability of the facts in such documents that our focus lies.

The elegant concept of public-key cryptosystems [12] and their implementation [20] enabled a content-dependent digital signature to be created for electronic documents (see Appendix A for an overview of digital signatures). Beth et al. [4] suggest that this changed the primary focus of the information security field from secrecy alone to broader notions of authentication, identification and integrity verification. With the steady rollout of Public Key Infrastructure (PKI), public, corporate and governmental acceptance of, and confidence in, digital signatures has steadily grown. Blakley posits that digital signatures are quite different from their ink-based predecessors and suggests that we should "look more closely at every way in which digital signatures differ" so that we may fully realize their worth [5]. We agree.

We are specifically interested in the technical constructs and mechanisms in a digital signature that afford the ability to selectively handle verifiable content securely and efficiently. Thus content extraction signatures (CES) [21] were developed to enable the signing of content at a granularity specified by the signer, rather than following the traditional practice of unconditionally signing at the container level (i.e. the whole document). An overview of CES is found in Appendix A.

Brands first proposed Digital Credentials in 1993 [7] and has further contributed extensive work towards en-

* Correponding author

hancing the privacy of Digital Credential holders in large part through anonymity and pseudoanonymity via the overhead of protocols and a Certification Authority [6, 8]. Some of these protocols provide the ability for the holder of a Digital Credential to selectively disclose a property of the attributes that has been encoded into the Digital Credential. In [8], Brands says of Digital Credentials, "Digital Credentials enable their holders to determine for themselves when, how, and to what extent information about them is revealed to others . . . ." While a Certification Authority issues the credentials for the owner's use, there appears to be no mechanism or policy for expressing restrictions on which information may be disclosed. It is simply left to the credential owner to decide. Depending on the use of the Digital Credential, there may arise situations for semantic abuse stemming from the ability to selectively disclose information.

Micali and Rivest introduced "transitive signature" schemes [17], and Bellare and Niven later presented performance improvements for such schemes [2]. Transitive signatures allow a signer to sign edges and nodes of a graph such that a signature for any edge in the transitive closure of the signed graph can be generated that is indistinguishable from the signature that would have been generated had the original signer signed that edge. This scheme shares with CES the notion of enabling valid signatures to be generated for transformations of an original signed object, though in this case the signatures are for information implicit in the original graph structure rather than extracted subsets of it. A general approach to homomorphic signature schemes for some binary operations has been reported by Johnson et al. [15].

The XML Signature (XMLsig) specification [1] is a joint proposal from the World Wide Web Consortium (W3C) [22] and the Internet Engineering Task Force (IETF) [14]. It defines a scheme for creating digital signatures that can be applied to digital content and that may be located internal to the document or externally on various sites across the Web. Whilst there are some similarities, or parallels, with CES, the XMLsig does not provide for the CES security for blinded content, nor does it permit a signer to specify an Extraction Policy.

Polivy and Tamassia [19] present an architecture for authenticating responses to queries from untrusted mirrors of authenticated dictionaries using Web services and XML Signatures. They also implement a custom XML Signature transform. In other work, Devanbu et al. have proposed a new approach to signing XML documents to enable certification of answers to arbitrary queries [11].

1.1 Motivating example

Digital libraries today often embrace a commercial model whereby articles, books etc. are available through various mechanisms such as subscriptions or ad hoc purchases. This information is handled at a container level where the entire container must be purchased as the user cannot simply purchase a page, or a section, from the paper. In addition, the information is not commonly signed so that the receiver can authenticate the content's source. The ability to authenticate information and discern its source is important these days since anybody can publish through Web pages, bypassing the traditional editorial/publishing process. If the user who purchases an article wants to use some of the content in a document of their own, there is little alternative to copying the content and then pasting it into the document (assuming an appropriate format) as well as entering the citation information.

Ideally, the user should be able to purchase and work with just the information they require. This information should be signed so that a reader of the work can verify and authenticate the content.

In the case of a digital library, a user should be able to retrieve either all of the signed collection of fragments (i.e. the entire article or book) or a signed subset of fragments. If the entire collection of signed fragments is retrieved, then the user should be able to use fragments at a later time as required. These fragments should be able to be verified and embedded in another document. Accompanying these fragments should be metadata that can be used to automatically add an entry to the bibliography if one is in use.

1.2 Contents of this paper

In this paper we introduce a new *Hierarchical Grouping Extraction Policy* for use with Content Extraction Signatures (CES). We demonstrate its implementation using XML Signatures and then illustrate enriched functionality for digital libraries through the use of CESs using the new grouping policy.

Section 2 gives the reader some background by introducing CES through a brief overview, along with two motivating examples involving the selective handling of verifiable content.

A recap of our previously introduced Extraction Policies including details of the *Grouping Extraction Policy* is presented to provide a foundation for introducing and presenting the framework for the new *Hierarchical Grouping Extraction Policy* in Sect. 3. Also included is a comparison of the various Extraction Policies and their implementation costs to assess the new scheme.

After giving a brief overview of XML Signatures, in Sect. 4 we show how to implement the new Hierarchical Grouping Extraction Policy and achieve CES functionality using the open standard XML Signature to enable development of interoperable applications. We also show an improved design for the XML Signature structure that enables it to handle Grouping Extraction Policies.

Having shown how to selectively handle verifiable content using CES, in Sect. 5 we conjecture as to how this may enrich the functionality of digital libraries in the emergent electronic society.

We close with some concluding remarks in Sect. 6.

2 Background

2.1 Content extraction signatures

Content extraction signatures (CES) were originally designed for use in multiparty interactions to overcome privacy concerns by enabling the selective disclosure of verifiable document content. CES permit the owner, Bob, of a document signed by a signer, Alice, to produce a subdocument (original document less some removed content) that can be verified by any third party, Carol, without revealing the contents of the removed portions of the original document [21]. The production of the sub-document may involve either *extracting* or *blinding* content from the original document. We discuss this further in Sect. 2.5 below.

To illustrate the use of CES, consider the typical example depicted in Fig. 1. Here we have the document signer Ace University, the document owner Bob (a student), and verifiers Carol and Don, who are potential employers. In this example, Ace University issues a student Bob with a formal document: an academic transcript (original document). Bob is required to include the formal document with a job application document sent to a prospective employer Carol. Note that the academic transcript document is likely to include Bob's personal details, for example his date of birth (DOB), etc. To avoid age-based discrimination, Bob might not wish to reveal his DOB to Carol (indeed, in some countries it is illegal for a prospective employer to seek the applicant's DOB). The university understands this and is willing to allow employers to verify academic transcripts with the DOB, and possibly with other fields, removed. The university, however, may require some fields to be included in every extracted document.

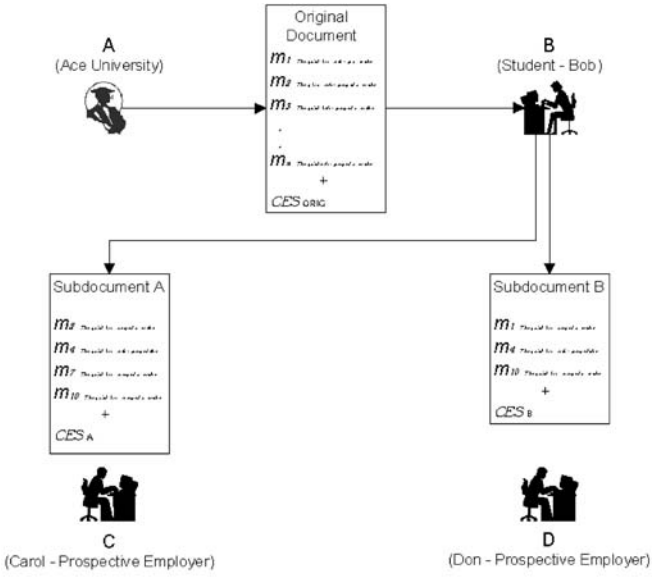

Fig. 1. A real-life scenario for selective disclosure

An essential and integral component of CES is the signer's Extraction Policy as it enables the signer to specify which fragments may be extracted or blinded. This affords protection from semantic abuse: abuse arising from the use of the content in an out-of-context manner. Extraction Policy validation is a requirement for CES validation.

In short, CES enable selective disclosure of verifiable content, provide security for blinded content through the use of a salt, and enable the signer to specify the content that the document owner is allowed to extract or blind. Combining these properties yield what we call *CES functionality.*

2.2 Bandwidth issue

The "maximally" coarse granularity of signed information using the standard digital signature causes unnecessary bandwidth usage. Consider Bob, the document owner, who wants to pass on a single item of verifiable information to Carol. Instead of being able to pass on this single piece of information, Bob is forced to furnish the entire document, which could be significantly greater in size than the single item; otherwise Carol will not be able to verify the signer's signature on the information.

To illustrate such a scenario that is not a privacy issue but one of information relevance, consider an electronically published article in which some aspect of an interview with the Prime Minister (PM) is reported. As depicted in Fig. 2, the PM's office issues a transcript of the interview involving the PM that has been signed using the standard digital signature.

The publisher would like to quote only the PM's response to a particular question since there are tight constraints on article size and it is neither appropriate nor possible to include the entire transcript of the interview.

It is highly desirable for the reader to be able to verify the quoted content in the article that originates from the signed interview transcript, since it would eliminate problems of misinterpretation and misquoting.

This example illustrates the trade-off that exists between verifiable content granularity and bandwidth, as illustrated in Fig. 3. This trade-off is likely to arise in many other scenarios as the Internet burgeons. A further goal of this work is to reduce the signed content granularity and move towards reduced bandwidth.

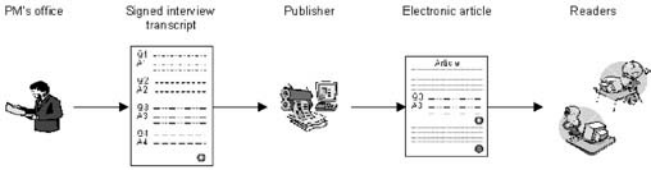

Fig. 2. Example of electronic publishing that includes verifiable content sourced from another signed document

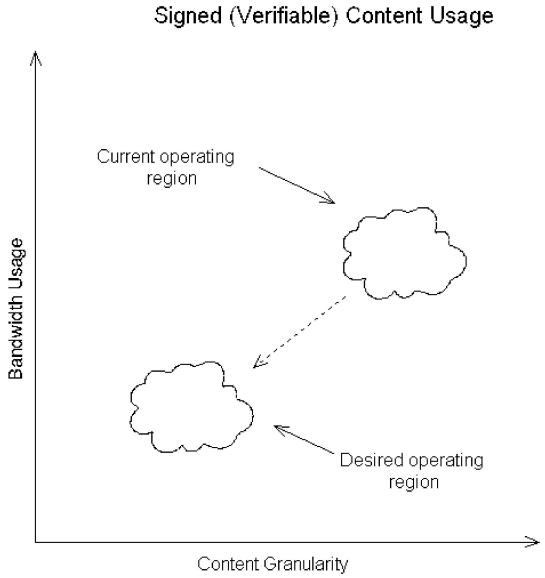

Fig. 3. Trade-off between signed content granularity and bandwidth usage. As the granularity of signed content increases, it results in increased bandwidth usage. Signed content is characterized by large grains of information resulting in high bandwidth requirements. Signing content in a smaller-grained manner results in lower bandwidth requirements

2.3 Selective content disclosure abuse

The ability to selectively disclose information contained in a document also has a potential risk, since the information accompanying a fragment in a document often provides the context. The disclosed fragment may have a different meaning when it is not accompanied by certain other information that is present in the original document.

For example, using the scenario depicted in Fig. 2, to avoid the PM's responses being quoted out of context, it is desirable that the question and the response be linked, so that the response is always preceded by the corresponding question. Hence there is a requirement that the signer be able to exert some control over what verifiable content can be selectively disclosed by the document holder. Conceivably, the document signer would want to be able to specify which fragments can:

- be extracted in isolation,
- be extracted only when accompanied by other specified fragments,
- be extracted optionally accompanying other specified fragments, and
- never be extracted (i.e. can only be provided with the entire document).

It is vitally important to protect against semantic abuse when providing the ability to selectively handle information. Therefore, the design of CES includes a signer-specified Extraction Policy that enables the signer to specify precisely the content that may be disclosed.

2.4 The orthodox approach

We now consider the feasibility of using the standard digital signature for selectively handling verifiable information using the academic transcript scenario depicted in Fig. 1 above.

Ace University could sign the individual fragments in Bob's academic transcript and forward them to Bob. This would entail n signatures for n fragments in the document. Bob could then forward only m appropriate fragments to the prospective employer, Carol. However, this would require n signing operations by Ace University and m signature verifications by Carol. Furthermore, Ace University *cannot* protect against semantic abuse stemming from selective disclosure.

An alternative approach is to decide upon allowed subsets of fragments corresponding to various permissible fragment groupings. Each of these subsets could then be signed and issued as a separate document. This approach has an upper bound of 2^n possible subsets that would entail 2^n signatures by Ace University and would present a considerable document management challenge for Bob. Using the standard digital signature in this manner departs from the conventional single-document orthodoxy and involves many signed documents. This would require significantly more storage space and complicate the handling required by Bob. Notwithstanding the storage problem, the prospect of searching through the collection of documents with various fragment combinations to find the required combination of fragments to disclose to Carol would be daunting. This approach is clearly infeasible.

2.5 User conceptual models

There are notionally two conceptual models that reflect the perspective of the document owner when selectively disclosing verifiable information. Each conceptual model represents one half of a range of 0–100% content disclosure. We call these conceptual models *blinding* and *extracting*. The adoption of either by the document owner is influenced by the requirements for the information to be disclosed.

The blinding model involves the disclosure of most of the original document as illustrated in the academic transcript example in Fig. 1 above, i.e. extracting most of the document content by blinding some content. Alternatively, the extracting model involves the disclosure of only a small amount of the document as illustrated in the electronic publishing example in Fig. 2 above, i.e. blinding most of the document by extracting only some content.

3 Extraction policies

The function of the Extraction Policy is not to enforce what content is disclosed. Instead, it specifies which subdocuments are permissible and an *extracted CES*, or *extracted signature*, can be generated. The extracted CES

enables the recipient of an extracted sub-document to verify the authenticity of the content. Indeed, the digital signature may verify the content integrity; however, this does not mean that the CES is verified.[1] To ensure semantic integrity, compliance with the signer's extraction policy is also required. Thus the CES verification algorithm not only involves verifying the document content, it also includes checking the fragments for compliance with the Extraction Policy.

An extracted CES is simply a mutation of the CES and does not involve any cryptographic operations. It still uses the same digital signature from the signer, and hence there is no requirement for the signer's secret key. See Appendix A.2 for further details.

The Extraction Policy is embodied as an encoding of all the allowed fragment extraction subsets in a structure called a *Content Extraction Access Structure* (*CEAS* for short). Thus the CEAS is an integral component of CES and is included as input to the signing and verification algorithms.

3.1 Single-dimensional policy

The single-dimensional Extraction Policy and a simple structure to support it, initially proposed with CES [21], will now be recapped.

Depending on the nature of the document and the content being signed, a very simple Extraction Policy may suffice. This includes content where there are no contextual semantics and hence no need to specify fragment grouping. The fragments are simply treated individually in a binary sense as being either *mandatory* or *optional*, where a mandatory fragment *must* be contained in the sub-document, whereas an optional fragment *may* be contained in the sub-document. Therefore, the Extraction Policy can be efficiently encoded using a single bit for each fragment.

The earlier example illustrated in Fig. 1 above, involving the student forwarding a signed electronic version of his/her academic transcript to a prospective employer, could involve a single-dimensional Extraction Policy. In this example the student wants to simply blind his or her date of birth in the transcript.

Single-dimensional Extraction Policies have very low implementation costs but do not support fragment grouping and, hence, are suitable where there are no contextual semantics for the fragments.

3.2 Richer multidimensional policies

Now we focus on Extraction Policies that will support the ability to select and extract fragment groupings as well as the ability to specify the fragment grouping relationships as being either mandatory or optional. Thus we now have a multidimensional view of the fragment.

This presents a challenge: how do we achieve this richness and flexibility in the Extraction Policy whilst constraining the size of the CEAS, which contains the encoding of this information, and hence the size of the extraction signature? The multidimensional policies described below will be treated according to the *extracting* conceptual model.

3.2.1 Grouping

We will now revisit in some detail the *Grouping Extraction Policy*, proposed in [9], to establish a foundation and framework for presenting a new Hierarchical Grouping Policy along with its encoding in the CEAS.

First we will redefine our fragment types used earlier for the Single-Dimensional Extraction Policy by replacing the *mandatory* and *optional* types with *primary* and *secondary* targets, respectively. A *primary* target fragment can be extracted *in its own right* from the original document to be a part of the sub-document. Only primary targets may be *directly* selected, or targeted, for extraction. If a fragment is not a primary target, then it is a *secondary* target. This means that it may be extracted only in association with a primary target fragment.

Fragment groupings are specified through the use of an association from one fragment to another fragment. A fragment may have zero or more associations with other fragments. Each association is either *mandatory* or *optional*, and all associations are asymmetric and transitive. Also, mandatory associations are relative to a primary target fragment and always subsume optional associations with respect to transitivity. If a fragment has a mandatory association with a primary target fragment, it means that the associated fragment *must* accompany the primary target if it is extracted. A fragment that has an optional association with a primary target fragment *may* accompany the primary target fragment if it is extracted. Associations are mutually exclusive since a fragment cannot have both a mandatory and an optional association with another fragment.

We will now describe fragment grouping options and their use by the document owner. A fragment type and its extraction permissions can be identified as:

- a primary target with no associations: it can be extracted by itself;
- a primary target with mandatory associations: if extracted it must be accompanied by its associated mandatory fragments;
- a primary target with optional associations: if extracted it may be accompanied by its associated optional fragments;
- a primary target with mandatory associations from *all* other primary targets: a mandatory fragment that must accompany any primary fragment that is extracted;

[1] This is not the case with CES implemented using XML signatures described in Sect. 4 as CES functionality is included in the XMLsig Core Validation process.

– a secondary target with no associations: it can never be extracted;
– a secondary target with mandatory associations: it can only be extracted when accompanying a primary target fragment via a mandatory association; or
– a secondary target with optional associations: it can only be extracted when accompanying a primary target fragment through an optional association.

CEAS using byte lists. A simple approach to storing the signer's fragment Extraction Policy is to use lists for the fragment associations. We implement for each fragment a list for either its mandatory or its optional associations.

A fragment's type is determined by whether or not its self-referent fragment number is contained in the list: primary target type if in the list, or secondary target type if not in the list.

The type of associations with the fragment numbers contained in the list is in turn determined by the fragment type: primary target lists define mandatory associations whilst secondary target lists define optional associations.

Using a 32-bit fragment identifier, the size of the CEAS for a document containing 200 fragments with a fragment association density of say 20% (i.e. an average of 40 associations per fragment) and a primary target density of say 50% (i.e. 100 of all the fragments are a primary target) would be 257.92 kbits.

CEAS using bit vectors. Bit vectors could be used as an alternative to using lists, where for a document with n fragments we allocate a vector of n bits for each fragment. This can be represented as an $n \times n$ bit matrix, irrespective of the number of associations. Since there are n bits available per fragment, we use:

– *the self-referent bit* to specify whether or not the fragment is a primary target or a secondary target; and
– *the non-self-referent bits (or other bits)* to specify the mandatory or optional fragment associations, of which there are $n-1$.

The type of association specified by the other bits depends on whether the fragment is a primary or secondary target. For primary targets the non-self-referent bits define the mandatory associations, while for secondary targets they define the optional associations. Also, there are no optional associations between two primary fragments. This would be redundant, since the two primary fragments can simply be extracted if required. See [9, Sect. 4.2] for a detailed explanation of an example of a CEAS encoding using a bit vector.

Practical example. To illustrate a scenario where a Grouping Extraction Policy would be used, consider the electronic publishing example discussed earlier in Sect. 2.2. In this case the Prime Minister's response to a particular question could be defined as a primary fragment with a mandatory association specified for the preceding question. If the response fragment was extracted, then the preceding question fragment must also accompany it for the extracted signature to be verifiable. Alternatively, the question fragment could be specified as a primary fragment with an optional association to the response fragment that would be specified as a secondary fragment. In this case, the response fragment could not be directly targeted for extraction. However, it could optionally accompany the question fragment.

Lists vs. vectors. List-based representations are more efficient when fragment association density (i.e. edges per node) is low, particularly for large numbers of fragments. The bit matrix will be more efficient when the association density is high.

Recall that n was defined as the number of fragments in a document. We now define s to be the size of the fragment identifier in bits, a_d the fragment association density and p_d be the primary fragment density. The size of the list encoding in bits is

$$ns(n-1)a_d + nsp_d, \tag{1}$$

while the matrix encoding is

$$n^2. \tag{2}$$

The matrix encoding will thus be more efficient when

$$ns(n-1)a_d + nsp_d > n^2, \tag{3}$$

that is, when

$$a_d > \frac{n}{s(n-1)} - \frac{p_d}{(n-1)}. \tag{4}$$

As n approaches infinity, the right side of the inequality approaches $1/s$:

$$a_d > \frac{1}{s}. \tag{5}$$

Therefore, when using a fragment identifier size of 32 bits, the matrix encoding will be more efficient when the fragment association density is greater than approximately 3%.

The number of fragment associations specified by a signer is dependent on the document type, the fragment content and the signer's view of how the fragments may be used etc., and hence the fragment association density is highly variable. In an implementation, the software would know the number of fragments defined and all of the fragment associations. Therefore, prior to signing, the software could decide an appropriate fragment identifier size and accordingly the smaller CEAS encoding method.

Thus, for comparison with the example in Sect. 3.2.1 above, also with 200 fragments, the matrix representation would occupy 40 kbits.

3.2.2 Hierarchical grouping

Whilst the *Grouping Extraction Policy* described in the previous section supports the grouping of fragments, it does not permit the sub-grouping of fragments. Nor is it ideal for use with signing hierarchical documents (most commonly comprised of volumes, chapters, and sections etc.) as well as documents that have hyperlinks such as Web pages, or more generally XML documents, likely to be encountered in digital libraries. These types of documents can typically have large numbers of fragments along with localized clustering of fragments associations. The *Grouping Extraction Policy* lacks the expressiveness and efficiency to support fragment sub-grouping and as a result treats all fragments at the same level, leaving it to the user to perform fragment sub-grouping manually. We will now present and discuss a new extraction policy to enable the sub-grouping of fragments so that fragments in the sub-group can be efficiently handled either as an entire set, or as allowed subsets: a *Hierarchical Grouping Policy.*

The same basic concepts and definitions for fragment types and their associations as defined for a Grouping Extraction Policy are retained, although we introduce a notion of locality, or scope. We will adjust the definitions for fragment type and associations, as well as introduce some restrictions for their use within a locality.

Let us consider a fragment of content, in this case comprised of three paragraphs of text. This fragment can be divided into three segments called sub-fragments, or child fragments, as illustrated in Fig. 4. Extending further, each of the child fragments could in turn be divided into segments, or sub-fragments, and so forth until the desired content granularity is achieved. From the child fragment's perspective, its parent fragment is the most immediate fragment that minimally contains all of the content for that child fragment. The child fragment's content is also part of the content for all of its ancestor fragments.

The child fragment's type and associations are now handled relative to its locality and are as follows:

- A child fragment's type can be either a primary or secondary target.
- A child fragment's associations are only relative to its sibling fragments.
- Child fragments as a collection inherit their parent's type.
- Child fragments as a collection inherit their parent's associations.

Sub-fragments can only be associated with other fragments that are not sibling fragments through their parent's associations with the other fragments.

Fragments that are secondary targets and have no associations with other fragments cannot have any child fragments. This is because the parent fragment can never be disclosed in a sub-document. Therefore, there is no need to define child fragments since, as a collection, they can never be disclosed because they inherit the parent's type and associations.

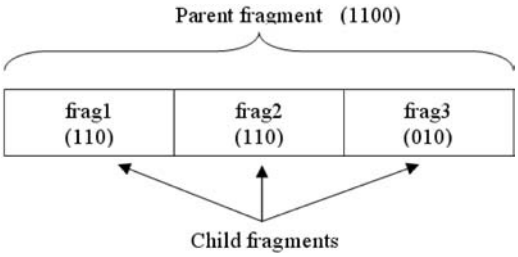

Fig. 4. Example of a parent fragment along with its extraction policy that has been segmented into three child fragments each with their own extraction policy

The child fragment's type and associations are first applied with respect to all the sibling fragments (i.e. within the scope of the parent fragment). Once this is complete, the collection of child fragments is then treated as a single item inheriting the parent fragment's type and associations. In turn, where the hierarchy extends to multiple levels, the parent node is treated along with its siblings in the same manner, repeating until the root fragment is reached.

Using this scheme, a collection of child fragments can be handled selectively. Alternatively, all of the child fragments can be handled collectively, treated as a single, albeit larger, fragment if required.

CEAS using byte lists. The simple approach described in Sect. 3.2.1 using lists is still applicable for storing the signer's fragment extraction policy. However, the notion of locality, or scope, is applied so that all fragment numbering with respect to the self-referent fragment number and fragment associations is relative to the child fragments of each parent fragment. Where there are multiple levels of sub-fragments, each parent fragment is in turn treated as a child fragment of its parent and so forth until the root fragment is reached.

CEAS using bit vectors. We use the same scheme detailed earlier in Sect. 3.2.1 for the Grouping Extraction Policy. However, we now use it within a context. Fragment numbering and fragment associations are treated in the same way as described above for byte lists.

Each fragment's vector size now changes from a fixed size of n bits for n fragments, to a varying size dependent on the number of sibling fragments it has. This means that the fragment vector size is not constant throughout the document, although it will be constant within each locality or collection of fragment child fragments.

A bit vector example explained. To illustrate the use of a Hierarchical Grouping Policy, consider a relatively simple document and its extraction policy encoding using bit vectors as denoted by the accompanying CEAS depicted in Fig. 5. This document has four main fragments, or highest level fragments, two of which are segmented

into three child fragments, or sub-fragments, each. The following is an interpretation of the signer's Extraction Policy for the document depicted in Fig. 5.

Frag1 is a secondary target and cannot be directly targeted for extraction. It can only be indirectly extracted through its mandatory association with frag2, or through its optional associations with frag3 and frag4.

Frag2 is a primary target that can be directly extracted. It also has a mandatory association with frag1 and an optional association with frag4. If frag2 is extracted, then it must be accompanied by frag1 and may be accompanied by frag4.

Frag2 is also a parent fragment because it has been segmented into three child fragments: frag2.1, frag2.2 and frag2.3. Frag2 can be handled as a single fragment that includes all of the child fragments, or as a collection of child fragments respecting the local fragment Extraction Policy. The local policy is as follows:

Frag2.1 is a primary target with a mandatory association with frag2.2. If frag2.1 is extracted, then it must be accompanied by frag2.2.

Frag2.2 is a primary target with a mandatory association with frag2.1 and an optional association with frag2.3. If frag2.2 is extracted, it must be accompanied by frag2.1 and may be accompanied by frag2.3.

Frag2.3 is a secondary target and can never be directly targeted for extraction. It can only be extracted through its optional association with frag2.2.

Frag3 is a primary target and, if it is extracted, it may be accompanied by frag1 through its optional association.

Frag4 is a secondary target and can only be extracted if it accompanies frag1 or frag2, through its optional association. Should this be the case, then it may also include frag1 through its own optional association.

Frag4 is also a parent fragment. The local policy for handling the child fragments is as follows:

Frag4.1 is a secondary target fragment and can only be extracted by accompanying frag4.2 through its mandatory association, or it may accompany frag4.3 through its optional association.

Frag4.2 is a primary target fragment and must be accompanied by frag4.1, while it may also be accompanied by frag4.3 through its optional association.

Frag4.3 cannot be directly extracted since it is a secondary target, although it may accompany frag4.1 or frag4.2.

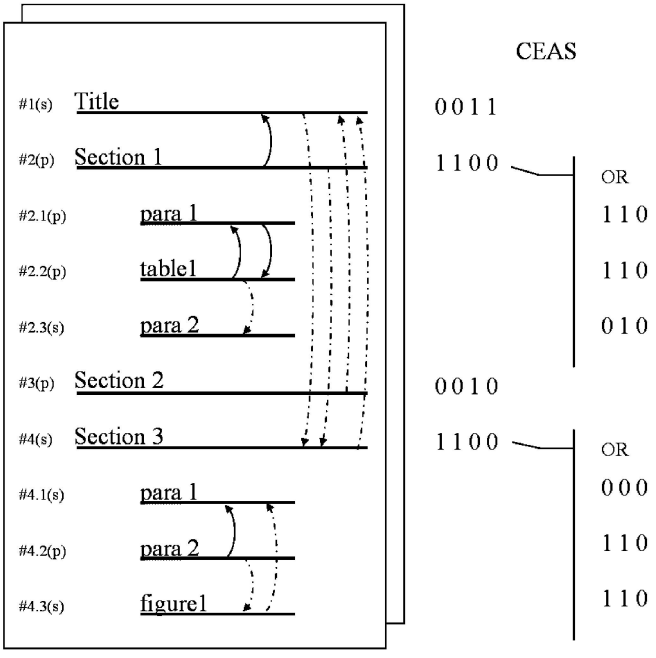

Fig. 5. Example of hierarchical grouping policy encoded using bit vectors and its mapping to a structured document showing four top-level fragments, two of which are parent fragments each with three child fragments

A practical example. Consider the document depicted in Fig. 5. If this is a journal article, a user may need all the material in Sect. 1 of the paper. Since Sect. 1 is contained in frag2, the user thus extracts frag2, which also includes child fragments 2.1, 2.2 and 2.3, along with a corresponding extracted CES so that the content can be verified. However, frag2 has a mandatory association with frag1 (the title); therefore, frag1 is also extracted to comply with the signer's Extraction Policy, thereby enabling the sub-document and the extracted CES to be verified.

Another user may simply require the information in Table 1 that is contained in frag2.2 of the paper. Frag2.2 is a primary target and is accordingly extracted along with frag2.1 due to its mandatory association. This association may have been specified as a mandatory association due to its description of the contents in the table. We do not want frag2.3, so we can ignore it since it is an optional association. Once the fragment Extraction Policy for the child locality has been respected, the parent's associations and type can be applied. This means that while frag4 may be optionally included, frag1 must accompany the extracted child fragments, resulting in the extraction of three fragments: frag1, frag2.1 and frag2.2.

Lists vs. vectors. To compare the size of each encoding scheme, we will now consider a document comprised of a shallow fragment structure similar to that depicted in Fig. 5 with an increased number of fragments. The document has 150 fragments and an Extraction Policy with the following characteristics:

- 30 top-level fragments:
  - 66.$\dot{6}$% are parent fragments,
  - 50% primary target density, and
  - 20% fragment association density.

Table 1. Comparison of CEAS encoding scheme sizes for each of the extraction policies. Superscripts (n) indicate derivation using Eq. n

CEAS encoding	Single dim.	Grouping	Hierarchical grouping
Byte list (kbits)	–	$145.44^{(1)}$	$17.57^{(13)}$
Bit vector (kbits)	0.15	$22.5^{(2)}$	$1.62^{(8)}$

– each parent has 6 child fragments:
  – 50% primary target density, and
  – 50% fragment association density.

The size of the CEAS using the bit vector encoding scheme is as follows:

$$1.62\,\text{kbits} = 30^2 + 20 * 6^2. \tag{6}$$

The size of the CEAS for the list encoding scheme, allowing 32 bits for the fragment identifier, is as follows:

$$17.568\,\text{kbits} = 32 * (30 * .5 + 30 * (30-1) * .2 + 120 * .5 + 120 * (6-1) * .5). \tag{7}$$

From this relatively straightforward example it can be seen that there is a significant difference between the costs of the two CEAS encoding schemes. This difference is apparent with the example containing just two levels of hierarchy: the difference increases as the hierarchy grows deeper.

Before discussing a more general consideration of size, we define the following variables:

s – size of fragment identifier in bits
n – number of fragments for generation i
ρ – parent density for generation i, i.e. percentage of fragments for generation i that have child fragments
p_d – primary target density for generation i
a_d – fragment association density for generation i
ϕ – average number of child fragments per parent for generation i
k – total number of generations

For the bit vector scheme the size is as follows:

$$n_0^2 + \sum_{i=1}^{k} (n_{i-1}\rho_{i-1}\phi_i^2). \tag{8}$$

For the byte list scheme the size is comprised of the following components:

$$\text{Size of Parent Primary Targets} = n_0 p_{d0}, \tag{9}$$
$$\text{Size of Parent Frag Asns} = n_0(n_0-1)a_{d0}, \tag{10}$$

regressively including the following generations:

$$\text{Size of Child Primary Targets} = (n_{i-1}\rho_{i-1}\phi_i)p_{di}, \tag{11}$$
$$\text{Size of Child Frag Asns} = (n_{i-1}\rho_{i-1}\phi_i)(\phi_i - 1)a_{di} \tag{12}$$

for a total size of:

$$s(n_0 p_{d0} + n_0(n_0-1)a_{d0} + \sum_{i=1}^{k}(n_{i-1}\rho_{i-1}\phi_i)p_{di} + (n_{i-1}\rho_{i-1}\phi_i)(\phi_i - 1)a_{di}). \tag{13}$$

3.3 Comparison of Extraction Policies

Recall that the list-based encoding was shown, in (5), to be more efficient than the bit vector approach for the *Grouping Extraction Policy* for low fragment association densities. This is also the case for the *Hierarchical Grouping Extraction Policy*, although the fragment association densities need to be much lower, particularly with documents that have many levels of hierarchy. As can be observed from Table 1, the *Hierarchical Grouping Extraction Policy* is significantly more efficient than the *Grouping Extraction Policy*.

3.4 Signing the document

Signing the document using CES includes specifying the Extraction Policy, which involves a two-step process: (i) define the fragments and then (ii) specify the fragment associations. The process of defining a fragment includes specifying the content itself as well as whether it is a primary or secondary target. Once all the fragments have been defined, the signer specifies the mandatory and optional fragment associations for each fragment. This information is included as part of the extraction signature. Upon completion of signing, the document and its extraction signature (if separate from the document) are forwarded to the document user.

4 Implementation using XML signatures

Content extraction signatures enable selective disclosure of verifiable content, provide privacy for blinded content through the use of a salt, and enable the signer to specify the content the document owner is allowed to extract or blind. When combined, these properties give what we call *CES functionality*.

To enable the development of interoperable applications using CES with the new Hierarchical Grouping Policy we will now show how to implement XML Signatures to achieve *CES functionality*. This is achieved through the use of a new enhanced custom transform and a redesigned XMLsig structure first introduced in [10].

4.1 XML Signatures in brief

An XMLsig is comprised of four main components or elements, as illustrated in Fig. 6. The `<SignedInfo>`

element includes all of the content or resources to be signed with each item having a corresponding <Reference> element that identifies the content and a digest over it. The <Reference> elements are digested and cryptographically signed in a manner similar to signing when using a standard digital signature. The resulting signature value is stored in the <SignatureValue> element. The <KeyInfo> and <Object> elements are optional.

4.1.1 The reference processing model

The signed content, which may be contained in the same document as the XMLsig and/or external to the document containing the XMLsig, is referenced with a <Reference> element. The URI (Uniform Resource Identifier) [3] attribute of the <Reference> element identifies the signed item. Each <Reference> element may have zero or more transforms, which are applied to the dereferenced content prior to its being digested using the algorithm specified in the <DigestMethod> element. The resulting digest is always base64 encoded [13] and stored in the <DigestValue> element.

The <Transforms> element may contain an ordered list of transforms to be applied to the dereferenced content. Each transform is specified using a <Transform> element as follows:

```
<Transforms>
  <Transform Algorithm="t1" />
  <Transform Algorithm="t2" />
  ...
  <Transform Algorithm="tn" />
</Transforms>
```

The XMLsig's Reference Processing model [1, Sect. 4.3.3.2] specifies that the dereferenced content is supplied to the first transform. As illustrated in Fig. 7, the list of transforms forms a transform chain where the output from the first transform is supplied as the input to the second transform, its output to the next, and so forth, until the last transform, the output of which is supplied to the digest algorithm. The types of transforms defined include: Canonicalization (with comments and without comments); Base64; XPath Filtering; XSLT; and Enveloped Signature transform. The XMLsig Reference Processing Model is also used for XMLsig reference validation [1, Sect. 3.2.1], which is a required part of XMLsig core validation.

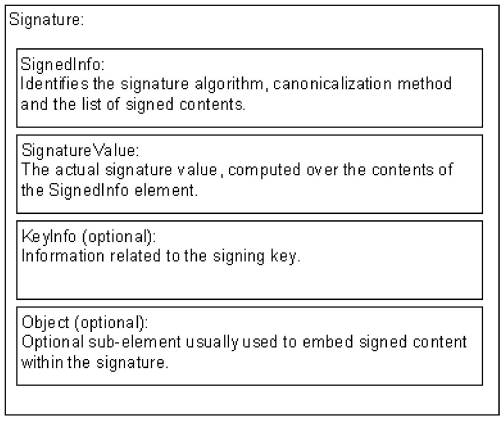

Fig. 6. Top-level components of an XML signature (adapted from [16, p.710])

Fig. 7. Transform chain for processing content prior to input to digest algorithm (adapted from [16, p.720]). The *arrows* indicate the transfer of data from one stage of the transform chain to the next

4.2 XML Signature design

As part of achieving CES functionality, compliance with the signer's Extraction Policy needs to be included in the XMLsig core validation [1, Sect. 3.2] processing requirements. This has been demonstrated in [10]; however, this was only with a simple, single-dimensional Extraction Policy. The policy-checking mechanism uses the Reference Processing model and is inserted into the <Reference> element being processed. Using this approach has the limitation that the transform chain is executed within a scope that is relative (and hence limited) to the current <Reference> element being processed. The problem with this is that to handle fragment grouping, the *VerifyPolicy* transform needs to have access to other <Reference> element contents that are effectively out of scope.

To solve this problem, the XMLsig needs to be restructured to enable the *VerifyPolicy* transform to access all of the fragment nodes. As illustrated in Fig. 8, this is achieved by making all of the <Fragment> elements children to the <Object> element and using a single <Reference> element to refer to the <Object> elements as follows:

```
<Reference URI="#obj1" Type=" ... #Object">
  <Transforms>
    <Transform Algorithm=" ... ces#VerifyPolicy" />
  </Transforms>
</Reference>
```

The <Object> element contains a <Fragment> element for each item that is to be signed as follows:

```
<Object Id="obj1">
  <Fragment Id="frag1" URI=" ... ">
    <CEAS type="LIST|VECTOR"> ... <CEAS>
    [<Salt> | <Digest>]
  </Fragment>
  <Fragment Id="frag2" URI=" ... ">
    <CEAS type="LIST|VECTOR"> ... <CEAS>
    [<Salt> | <Digest>]
  </Fragment>
```

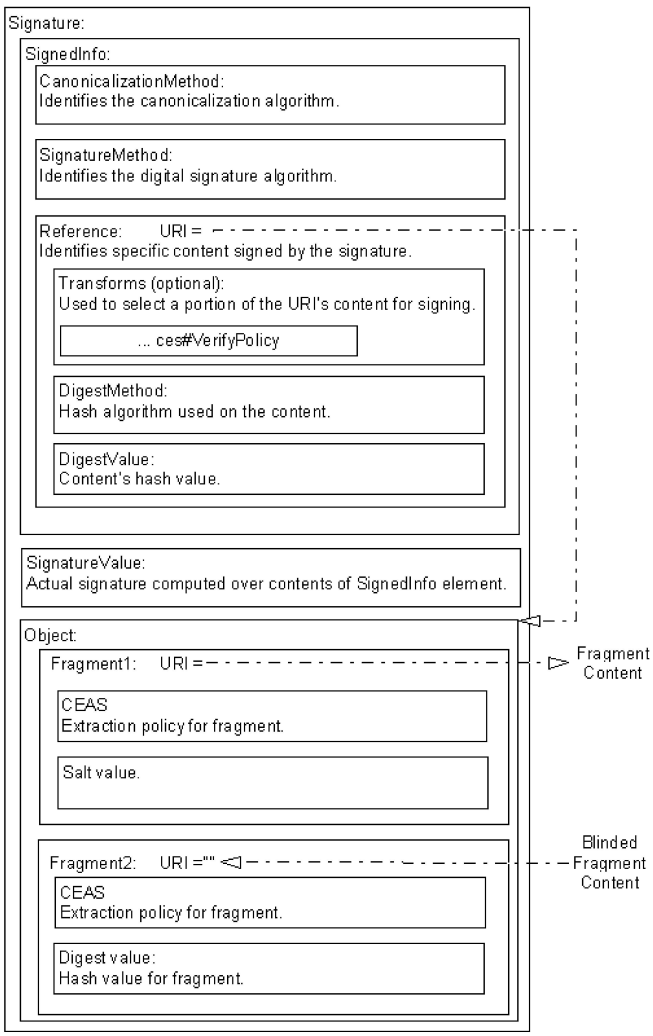

Fig. 8. XML signature structure for an extracted signature to bring all of the `<Fragment>` elements within the scope of the `<Reference>` element's custom transform in order to verify extraction policy compliance. Fragment1 is present and contains a URI attribute that points to the fragment content, as well as a Salt. Fragment2, on the other hand, has been blinded and contains a blank URI attribute while the salt has been mutated to a digest value for the content plus the salt

```
    ...
  </Object>
```

where | denotes an exclusive OR.

The URI attribute of the fragment references the fragment content while the `<CEAS>` element contains the encoding of the signer's Extraction Policy for that fragment. The `<Salt>` element contains a salt value used in CES to ensure privacy of blinded content [21, Sect. 3.3]. It is appended to the fragment content prior to digesting. The `<Salt>` element is always present in the original signature from the document signer.

When Bob, the document user, produces a sub-document, an extracted signature corresponding to the sub-document must be generated so that it can be validated by Carol, the sub-document recipient (or verifier), as being signed by Alice. This extracted signature has the `<Salt>` element replaced with a `<Digest>` element for the corresponding fragments that are not included (blinded) in the sub-document. The digest value is generated from the fragment content with the salt appended. Therefore, the extracted signature that is generated for the sub-document has a `<Salt>` or a `<Digest>` element for each fragment that is present or has been blinded as depicted in Fig. 8 for `<Fragment1>` and `<Fragment2>` respectively.

4.3 Custom Extraction Policy transform

The custom transform to verify the Extraction Policy used in [10] needs to be enhanced to handle the Hierarchical Grouping Extraction Policy. The URIs of custom transforms can be signed, as can the transform code itself, to establish trust. The requirement for the custom transform is to process the `<Reference>` element's dereferenced content by dereferencing the content of each of the `<Fragment>` elements and checking compliance with the Extraction Policy, and finally emitting a Result byte stream for input to the digest algorithm.

As illustrated in Fig. 9, the transform processes the dereferenced content from the `<Reference>` element that will be XML content containing at least one `<Fragment>` element. For each `<Fragment>` element the transform first checks for the presence of a `<Digest>` element that indicates the fragment has been blinded. If the fragment has been blinded, then the CEAS is checked for compliance with the fragment Extraction Policy. Compliance sees the CEAS appended to the digest value that is then appended to the Result. This Result will be emitted upon completion of processing of the last `<Fragment>` element. Should verification of the Extraction Policy fail, then two bytes of zeroes will be appended to the Result in place of the digest value. This will ultimately cause reference validation failure and in turn core validation failure since the appended bytes will not match those originally used to create the digest value stored in the `<Reference>` element's `<DigestValue>` element when it was signed.

On the other hand, if the fragment has not been blinded, the `<Digest>` element will not be found. Rather, a `<Salt>` element will be present. The fragment URI is dereferenced to retrieve the fragment content and the salt value from the `<Salt>` element is appended to the fragment content prior to digesting. The resulting digest has the CEAS appended to it and is then appended to the Result that will be emitted.

In addition to the explicit requirements of the transform, more subtly, it also accommodates the mutation of the `<Fragment>` elements, i.e. present fragments to blinded fragments. Normally the content referenced by a `<Reference>` element is invariant and a digest over it is included in the content signed by the cryptographic signature.

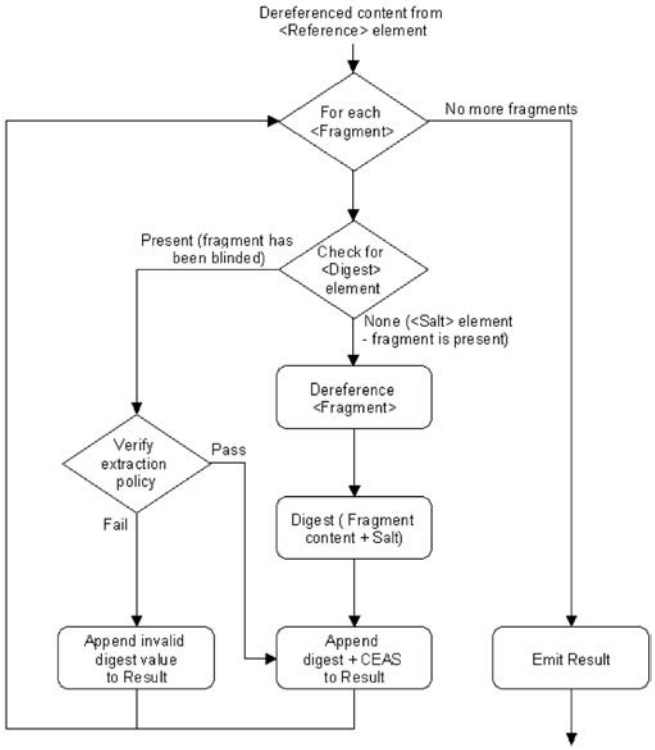

Fig. 9. *VerifyPolicy* transform algorithm for fragment grouping

5 Enriching digital library functionality

Having demonstrated the technical feasibility of selectively handling verifiable information and showing how to implement the Grouping Extraction Policy for CES using the XML signature open standard, we would now like to turn to digital libraries in a future electronic society. We will use a typical example from academia to briefly illustrate a facet of our conjecture, ignoring any economic model that is likely to accompany an actual deployment.

5.1 A specific example

Consider a researcher who is writing a paper and wishes to cite some other person's work that has been published and stored in a digital library. The material to be cited is contained in a published paper that has been signed by the publisher using a content extraction signature. With a suitable application, the publisher makes the entire paper, or fragments thereof, available for download. In this case the researcher selects the required content and extracts it along with an extracted CES. The extraction process is inexpensive in CPU terms because it does not include any cryptographic operation. If the researcher has a local copy of the paper, the extraction is simply performed locally.

The content fragments along with the extracted CES are embedded into the researcher's paper. The in-text citation is coloured:

- green to indicate the content that it anchors has been verified,
- red to indicate the anchored content has failed verification, or
- black to indicate that verification has not yet been performed.

In addition, hovering the mouse pointer over the in-text citation displays, through a pop-up window, the content to which the citation refers for the convenience of the reader. The embedding process also automatically inserts an entry into the list of references at the end of the document using the metadata that accompanies the embedded fragments.

The embedding of the content fragments from the referenced document into the researcher's paper makes the specific content, not the entire document, immediately available to the reader. The reader can have a high degree of confidence about the referenced material since the content is protected by a digital signature and upon verification can be certain that it has not been altered. In addition, the source is authenticated by the digital signature, thus enabling the reader to determine the veracity of the referenced content through the authority and reputation of its source.

This example represents just one possibility arising from the ability to selectively handle verifiable information in an electronic society. There may exist many other scenarios such as Web portals that aggregate information from multiple sources, or multiparty business interactions/transactions where minimal disclosure of information to various parties is required, etc.

6 Conclusion

We have shown the trade-off between verifiable content granularity and bandwidth usage, along with the importance of protecting against selective disclosure abuse, or semantic abuse. Responding to these types of emerging need, the development of content extraction signatures enables content to be signed in a finer-grained manner. We have also demonstrated an Extraction Policy that specifies which content can be verified when selectively disclosed.

After revisiting previous work on Extraction Policies to establish a framework upon which to build, we presented a new, richer and more efficient policy called a *Hierarchical Grouping Policy*. The new Extraction Policy is particularly suited for use with hierarchical documents such as journals, journal articles, and encyclopædias – not to mention the HTML, and increasingly XML, documents almost ubiquitous in modern electronic repositories.

We then showed how to implement CES with the new extraction policy using XML signatures, along with a new custom transform and improved XML signature structure to handle Grouping Extraction Policies.

After establishing the technical feasibility of handling verifiable content in a fine-grained manner, we offered an example of its potential use to enhance the functionality of digital libraries in an emergent electronic society.

Acknowledgements. The authors would like to thank Mr. Gerry Butler for his valuable comments and discussion during the development of this work.

Also, we wish to thank the anonymous reviewers for their constructive, detailed comments and suggestions in helping to improve the correctness and presentation of this paper.

References

1. Bartel M, Boyer J, Fox B, LaMacchia B, Simon E (2002) XML-signature syntax and processing. In: Eastlake D, Reagle J, Solo D (eds) W3C Recommendation. World Wide Web Consortium, 12 February 2002. http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/. Last accessed: 21 May 2004
2. Bellare M, Neven G (2002) Transitive signatures based on factoring and RSA. In: Zheng Y (ed) Proc. 8th international conference on the theory and application of cryptology and information security (ASIACRYPT 2002), Queenstown, New Zealand, 1–5 December 2002. Lecture notes in computer science, vol 2501. Springer, Berlin Heidelberg New York, pp 397–414
3. Berners-Lee T, Fielding R, Masinter L (1998) RFC 2396. Uniform resource identifiers (URI): generic syntax. Available online, August 1998. http://www.ietf.org/rfc/rfc2396.txt. Last accessed: 21 May 2004
4. Beth T, Frisch M, Simmons GJ (eds) (1992) In: Public-key cryptography: state of the art and future directions, 3–6 July 1992. EISS Workshop Oberwolfach Final Report. Lecture notes in computer science, vol 578. Springer, Berlin Heidelberg New York
5. Blakley GR (1999) Twenty years of cryptography in the open literature. In: Proc. 1999 IEEE symposium on security and privacy, Oakland, CA, 9–12 May 1999. IEEE Press, New York, pp 106–107
6. Brands SA (2000) Rethinking public key infrastructures and digital certificates: building in privacy. MIT Press, Cambridge, MA
7. Brands S (1993) Privacy-protected transfer of electronic information. U.S. Patent serial no. 5,604,805, February 1997, August 1993. Filed August
8. Brands S (2002) A technical overview of digital credentials. http://www.credentica.com/technology/overview.pdf. Last accessed: 18 February 2003
9. Bull L, McG Squire D, Newmarch J, Zheng Y (2003) Grouping verifiable content for selective disclosure. In: Safavi-Naini R, Seberry J (eds) Proc. 8th Australasian conference on information security and privacy (ACISP 2003), Wollongong, Australia, 9–11 July 2003. Lecture notes in computer science, vol 2727. Springer, Berlin Heidelberg New York, pp 1–12
10. Bull L, Stanski P, McG Squire D (2003) Content extraction signatures using XML digital signatures and custom transforms on-demand. In: Proc. 12th international World Wide Web conference (WWW2003), Budapest, Hungary, 20–24 May 2003. ACM Press, New York, pp 170–177. http://www2003.org/cdrom/papers/refereed/p838/p838-bull.html. Last accessed: 21 May, 2004
11. Devanbu PT, Gertz M, Kwong A, Martel C, Nuckolls G, Stubblebine SG (2001) Flexible authentication of XML documents. In: Proc. 8th ACM conference on computer and communications security, Philadelphia. ACM Press, New York, pp 136–45
12. Diffie W, Hellman ME (1976) New directions in cryptography. IEEE Trans Inf Theory IT-22(6):644–54
13. Freed N, Borenstein N (1996) Multipurpose Internet mail extensions (MIME). I. Format of Internet message bodies. Available online, 1996. http://www.ietf.org/rfc/rfc2045.txt. Last accessed: 21 May 2004
14. IETF (2004) The Internet Engineering Task Force. http://www.ietf.org/. Last accessed: 21 May 2004
15. Johnson R, Molnar D, Song D, Wagner D (2002) Homomorphic signature schemes. In: Preneel B (ed) Topics in Cryptology – CT-RSA 2002: the cryptographer's track at the RSA conference 2002, San Jose, CA, 18–22 February 2002. Lecture notes in computer science, vol 2271. Springer, Berlin Heidelberg New York, pp 244–62
16. LaMacchia B, Lange S, Lyons M, Martin R, Price K (2002) .NET framework security. Addison-Wesley, Boston, MA
17. Micali S, Rivest RL (2002) Transitive signature schemes. In: Preneel B (ed) Topics in Cryptology – CT-RSA 2002: the cryptographer's track at the RSA Conference 2002, San Jose, CA, 18–22 February. Lecture notes in computer science, vol 2271. Springer, Berlin Heidelberg New York, pp 236–243
18. NIST (1994) Digital Signature Standard (DSS). Number 186 in Federal Information Processing Standards publication. National Institute of Standards and Technology, May 1994. http://www.itl.nist.gov/fipspubs/fip186.htm. Last accessed: 7 April 2004
19. Polivy DJ, Tamassia R (2002) Authenticating distributed data using web services and XML signatures. In: Proc. 2002 ACM workshop on XML security (XMLSEC-02), New York, 22 November 2002. ACM Press, New York, pp 80–89
20. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2):120–8
21. Steinfeld R, Bull L, Zheng Y (2001) Content extraction signatures. In: Kim K (ed) Proc. 4th international conference on information security and cryptology (ICISC 2001), Seoul, Korea, 6–7 December 2001. Lecture notes in computer science, vol 2288. Springer, Berlin Heidelberg New York, pp 285–304
22. World Wide Web Consortium (2004) The World Wide Web Consortium. Available online. http://www.w3.org/ Last accessed: 21 May 2004

Appendix : Standard digital signatures and content extraction signatures

For the reader unfamiliar with the details and workings of digital signatures, we first provide a brief overview of the standard digital signature. This also serves as a basis for the following overview of CES schemes *CommitVector* and *HashTree* [21, Sect. 4.1]. These two schemes are more computationally efficient than the third and fourth schemes proposed in [21, Sect. 4.2].

A.1 Standard digital signature

The Digital Signature Standard (DSS) [18] uses hashing and asymmetric encryption to produce a digital signature over the content that is to be signed. Signature generation and signature verification processes are used to produce and verify a digital signature respectively.

The generation process uses a hash function h on the content being signed M to produce a message digest H that is then encrypted with the function $\mathsf{Encrypt}$ using the secret key SK of the signer to produce a digital signature σ as follows:

$$\sigma = \mathsf{Encrypt}_{SK}(h(M)). \tag{A.1}$$

The verification process outputs a verification decision $d \in \{Accept, Reject\}$ following its processing of σ and the received content M'. Verification proceeds by hashing M' to produce H' and then comparing it with the result of decrypting σ with the function Decrypt using the signer's public key PK as follows:

$$\begin{aligned} &\text{If } h(M') = \mathsf{Decrypt}_{PK}(\sigma) = h(M) \\ &\text{then } M' = M \quad \text{(with extremely high confidence)} \\ &\quad \text{so } d = \text{“Accept”} \end{aligned} \tag{A.2}$$

A.2 CES-CommitVector scheme

The standard digital signature scheme encrypts a digest of the whole message. In contrast, for CES to facilitate the selective handling of verifiable information, we sign a collection of n fragment digests. We now consider the content to be signed as being comprised of content fragments

$$M = (m_1, m_2, \ldots, m_n)$$

To assist in understanding CES we will illustrate the following description with a practical example. This example involves an original document that contains $n = 5$ fragments and is signed with a CES by the document creator and then forwarded to the document owner. The document owner removes the third and fourth fragments to create a sub-document, containing the first, second and fifth fragments, and then generates an extracted signature for the sub-document. The sub-document along with its extracted signature are then forwarded to the verifier.

To support the handling of individual fragments, this CES scheme requires the storage of the message digests for any removed fragments. Therefore, to provide privacy security against leakage of information for removed fragments,[2] prior to hashing each fragment m_i a random value, or *salt*, r_i is appended as follows:

$$H_i = h(m_i + r_i). \tag{A.3}$$

The fragment message digests along with the encoding of the signer's Extraction Policy $CEAS$ are encrypted with the Encrypt function using the signer's secret key SK as follows:

$$\sigma = \mathsf{Encrypt}_{SK}(H_1, H_2, \ldots, H_5, CEAS). \tag{A.4}$$

[2] To illustrate the need for privacy security for removed fragments, consider the scenario depicted in Fig. 1, where a fragment could contain the grade a student received for a unit of study. Given that the grade may have only four possible values, it is trivial to determine the grade from the digest value by hashing each of the values and comparing each with the digest value. Likewise, a fragment containing a date of birth would not entail a large search space and could therefore be recovered from its digest value relatively easily.

The process of removing fragments by the document owner requires the generation of an extracted signature σ_{ext} to enable the verifier to verify the received fragments (or sub-document). To enable the signature extraction and verification processes, we must also include the salt values as well as the $CEAS$ in the signature σ_{CES} as follows:

$$\sigma_{CES} = (\sigma, r_1, r_2, \ldots, r_5, CEAS). \tag{A.5}$$

The document owner creates a sub-document to be passed on to a verifier by deleting the unwanted fragments, three and four in this case, and then generating the extracted signature σ_{ext} for the sub-document.

Where fragments are not removed, the message digest is not required; instead the salt value r_i is retained. This is because the digest can be generated during signature verification by hashing the received content m'_i after appending r_i as follows:

$$H'_i = h(m'_i + r_i). \tag{A.6}$$

However, where fragments are deleted, the fragment message digest must be retained to enable verification of the sub-document by its recipient, the verifier. Retaining the fragment message digest for each deleted fragment makes the salt value for the deleted fragment redundant. Therefore, the salt value is replaced with the fragment message digest for each deleted fragment. Thus the extracted signature generated by the document owner for the sub-document containing the first, second and fifth fragments is as follows:

$$\sigma_{ext} = (\sigma, r_1, r_2, H_3, H_4, r_5, CEAS). \tag{A.7}$$

The recipient verifies the sub-document

$$M' = (m'_1, m'_2, m'_5)$$

using σ_{ext}. First, the fragment message digests H'_i for the received fragments are generated by appending the salt values from σ_{ext} to the fragment as per (A.6). This gives the verifier H'_1, H'_2, H'_5 and $H_3, H_4, CEAS$ from σ, permitting a comparison with the result of decrypting σ with the Decrypt function using the signer's public key PK as follows:

$$\begin{aligned} &d = \text{“Accept”} \text{ if and only if} \\ &(H'_1, H'_2, H_3, H_4, H'_5, CEAS) = \mathsf{Decrypt}_{PK}(\sigma) \end{aligned} \tag{A.8}$$

However, before a CES verification decision d_{CES} can be issued, Extraction Policy compliance $(m'_1, m'_2, m'_5) \in CEAS$ must be checked. Therefore, we output the CES verification decision for our sub-document as follows:

$$\begin{aligned} &d_{CES} = \text{“Accept”} \text{ if and only if} \\ &d = \text{“}Accept\text{”} \; and \; (m'_1, m'_2, m'_5) \in CEAS \end{aligned} \tag{A.9}$$

A.3 CES-HashTree scheme

The *HashTree* scheme is a variant of the *CommitVector* scheme that uses a binary hash tree for handling the message digests for removed fragments. This scheme has the potential to substantially reduce the size of extracted signatures depending on the ratio of $n : m$, where n is the number of fragments in the original document and m is the number of fragments in the sub-document.

For this approach we use a binary hash tree with the leaf nodes containing the fragment message digests H_i. Each non-leaf node contains a hash value of the concatenation of its two child nodes, up to the root node H_{root}. Thus for CES signing, the encryption with the Encrypt function using the signer's secret key changes (A.4) to:

$$\sigma = \mathsf{Encrypt}_{SK}(H_{root}, CEAS). \tag{A.10}$$

while leaving (A.5) unchanged.

In the case of the extracted signature, using a hash tree obviates the need to store all of the fragment message digests for the deleted fragments. Instead, we need only keep the minimal number of fragment message digests and hash values for the internal tree nodes required to compute the root node H_{root}.

Following the development in [21, Sect. 4.3], we note that the length of the σ_{CES} using the *HashTree* scheme is the same as the length of the signature using the *CommitVector* scheme. It is with the extracted signature σ_{ext} where the difference is manifest. From [21, Sect. 4.3] we have the length of the extracted signature for the *CommitVector* and *HashTree* schemes as follows:

$$l_{\sigma_{extCV}} = l_\sigma + (n-m)l_H + ml_r + l_{CEAS} \tag{A.11}$$

$$l_{\sigma_{extHT}} = l_\sigma + f(n,m)l_H + ml_r + l_{CEAS} \tag{A.12}$$

where[3]

l_σ – length of signature obtained by encrypting the original document message digests and CEAS
l_H – digest length
l_r – salt length
l_{CEAS} – CEAS encoding length
$f(n,m) \stackrel{\text{def}}{=} min(m\log_2(n/m), n-m)$
n – fragment count in original document
m – fragment count in extracted sub-document

Whenever $n - m < n/2$, the *HashTree* extracted signature is shorter since $m\log_2(n/m) < n-m$. When $m \ll n$, the difference is dramatic. To illustrate with a practical digital library example, consider an encyclopædia with 100,000 entries, each of which is treated as a fragment. We wish to extract a single, verifiable entry. The length of the σ_{ext} component dependent on hashes would be $99,999l_H$ for the *CommitVector* scheme and $\log_2(100,000)l_H \approx 17l_H$ for the *HashTree* scheme. The *HashTree* scheme is thus overwhelmingly preferable when a small number of fragments is extracted from a large numbers of fragments in the original document to produce a sub-document, i.e. $m \ll n$.

In practice, the application software used in the fragment extraction process could simply decide which scheme to use based on the values for n and m prior to generating the extraction signature.

[3] $f(n,m)$ is the minimum of the worst case number of paths to H_{root} required to compute intermediate hashes from the remaining message fragments and hashes sent, or simply the same as for the *CommitVector* scheme.