Content Extraction Signatures using XML Digital Signatures and Custom Transforms On-Demand

Laurence Bull
School of Computer Science and Software Engineering, Monash University
Melbourne, Australia
Laurence.Bull@csse.monash.edu.au

Peter Stanski
School of Computer Science and Software Engineering, Monash University
Melbourne, Australia
Peter.Stanski@csse.monash.edu.au

David McG. Squire
School of Computer Science and Software Engineering, Monash University
Melbourne, Australia
David.Squire@csse.monash.edu.au

ABSTRACT

Content Extraction Signatures (CES) enable selective disclosure of verifiable content, provide privacy for blinded content, and enable the signer to specify the content the document owner is allowed to extract or blind. Combined, these properties give what we call *CES functionality*. In this paper we describe our work in developing custom transform algorithms to expand the functionality of an XML Signature to include *CES functionality* in XML Signature Core Validation.

We also describe a custom revocation mechanism and our implementation for non-XML content where the custom transforms are dynamically loaded demonstrating that custom signing and verification is not constrained to a 'closed system'. Through the use of dynamic loading we show that a verifier can still verify an XML Signature-compliant signature even though a custom signature was produced.

Categories and Subject Descriptors

K.4.1 [**Computing Milieux**]: Computers and Society—*Public Policy Issues*; K.4.4 [**Computing Milieux**]: Computers and Society—*Electronic Commerce*

General Terms

Security

Keywords

XML Signatures; Content Extraction Signatures; XML Signature Custom Transforms; Dynamic Signature Verification; .Net Framework XML Signature API

1. INTRODUCTION

The Internet continues to become more pervasive, affecting all areas of our lives. As the electronic society emerges, traditional commerce and interaction based on paper documents are becoming outdated and superseded by electronically-based processes and content. Digital signatures have enabled commerce and interaction in the burgeoning virtual world through the vital process of establishing trust.

While the use of digital signatures can help establish trust in electronic interactions, there can be a cost to privacy in some situations. These situations include multiparty interactions where there is some transfer of information from party A to party B who then forwards on some of the information to party C. To illustrate, consider a student applying for a job who wants to disclose some details from his/her academic transcript, which has been signed with a digital signature, without revealing his or her age.

Further, there may be other situations where the information flow from party A to party B is very large and the information flow from party B to party C is quite small. It is inefficient or inappropriate for party B to forward all of the original information from party A. An example of this would include a news article where the journalist cites an answer from a transcript of an interview with the Prime Minister, which has been signed with a digital signature.

Content Extraction Signatures (CES) [20] were developed in response to the need described above. CES 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. Combined, these properties give what we call *CES functionality*.

In recent years XML (Extensible Markup Language) has grown popular and enjoyed widespread deployment around the world. The joint work of the World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF) resulted in the XML-Signature Syntax and Processing Recommendation (XMLdsig) being published on February 12, 2002 [2]. This specification is believed to be sufficient to enable the independent development of interoperable applications using (XMLdsig).

In this paper we describe how we address the privacy and bandwidth issues in multiparty interactions by expanding the functionality of XML Signatures to include *CES functionality* within the XMLdsig specification. Our goal is to be able to achieve XMLdsig Core Validation [2, Sect.3.2] with this functionality using an XMLdsig compliant client. This is achieved through the use of a custom transform algorithm. We discuss our implementation, which is for non-XML content, and describe how we achieve dynamic loading of the transforms for signing and verifying documents. Further exploiting the custom transform approach we discuss our design of an additional transform for a revocation mechanism. We finally provide a brief description of the practical use of Content Extraction Signatures.

For this paper, the word 'fact' will be used in a general sense for a fragment of content that is verifiable. That is, it can be verified that the signer has issued and signed this content, regardless of whether the content is in any sense "true".

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005.

1.1 Contents of this paper

A brief review of Content Extraction Signatures along with a concrete example to illustrate the problem is provided in Section 2. We then introduce XML Signatures with an overview of their structure and then some detail about the Reference Processing Model. In Section 3, we discuss the XMLdsig structure for our design to cater for our custom transforms, followed by a description of our design for these transforms. Our implementation for non-XML content and the use of dynamic loading of the custom transforms is followed with some reflections on the implementation in Section 4. This is followed, in Section 5, with a brief description of the roles of users in the practical use of Content Extraction Signatures in our example setting. We close with some comments in Section 6.

2. BACKGROUND

Since the release of the XMLdsig specification W3C has released the Exclusive XML Canonicalization specification [6] which may be used as a transform with XMLdsigs. In addition W3C has released a newer and updated version 2.0 of the XML-Signature XPath Filter [7]. The authors are not aware of other work involving the development of custom transforms to expand the functionality of the XMLdsig or dynamically load transforms to suit particular functionality or requirements.

2.1 The problem

In order to clarify the issues described above, and our solutions to them, we will make use of a concrete example (adapted from [20]) involving a university, a student, and a prospective employer. This scenario results from discussions with an Australian university, hereafter called Ace University.

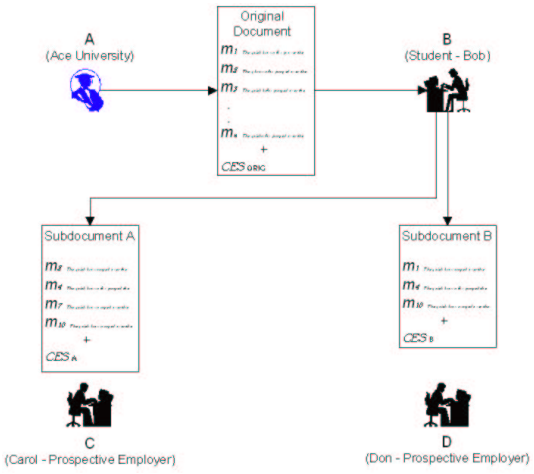

Figure 1: A real-life scenario for selective disclosure.

Consider the commonplace, although pertinent, example illustrated in Fig. 1. In this example, Ace University (A) issues a student Bob (B) 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 (C). Note that the Academic Transcript document is likely to include the Bob's personal details, for example, his date of birth (DOB). 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 removed (and possibly with other fields agreed to by the university removed as well, but not others which the university may require to be included in any extracted document). Yet if the university signs the Academic Transcript using a standard digital signature Bob must send the whole document to Carol to allow her to verify it.

The standard digital signature is well suited for point-to-point interactions as Ace University simply signs the content it wishes to send to Bob. However, not all instances of commerce can be characterised merely as a series of point-to-point interactions. Many instances are multipoint, or multiparty, interactions, where information flows from the information signer to the document holder and then to the information consumer(s) as illustrated in Fig. 1. It is this type of interaction that we address. The problem in this situation is that Ace University signs the information that it is sending to Bob, thereby requiring Bob to disclose all of this information in his interaction with Carol: otherwise it will fail verification. This occurs even though only part of the information is required for the interaction and Bob might not want to reveal all of the information.

2.2 Content Extraction Signatures

Content Extraction Signatures (CES) [20] are designed for use in multiparty interactions to overcome privacy concerns by enabling the selective disclosure of verifiable document content. Permitting the owner, Bob, of a document signed by Ace University, to produce an 'extracted signature' for an extracted subdocument (original document less some removed, or "blinded", content), which can be verified (to originate from Ace University) by any third party Carol, without the knowledge of the unextracted (blinded) document content.

There are notionally two modes of use: blinding facts (extracting most of the document content and blinding some content), or extracting specific facts (blinding most of the content and extracting only some content) [10]. Each mode reflects the perspective of the document owner when selecting content for disclosure and represents each end of a continuum.

An integral component of CES is the signer's *Extraction Policy*, which enables the signer to specify which facts may be extracted or blinded. This affords protection from abuse arising from the use of the facts in an out of context manner. Extraction Policy validation is a requirement for CES validation.

Originally four schemes were proposed for CES along with an assessment of their computation and bandwidth costs compared to signing using a standard digital signature [20, §4.3]. The work being reported in this paper involves the implementation of the first scheme, the *CommitVector* scheme, and its application using XMLdsig.

2.3 XML Digital Signatures

Basically, an XMLdsig is comprised of four main components or elements: `<SignedInfo>`, `<SignatureValue>`, `<KeyInfo>` and `<Object>`. The `<SignedInfo>` element includes all of the content or resources to be signed with each item having a corresponding `<Reference>` element, which 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.

An XMLdsig has the `<Signature>` element as the root element for its XML tree. It contains the four main components and has the following generic structure as defined in the specification [2]:

```
<Signature>
```

```
<SignedInfo>
  <CanonicalizationMethod />
  <SignatureMethod />
  (<Reference>
    (<Transforms>)?
    <DigestMethod>
    <DigestValue>
  </Reference>)+
</SignedInfo>
<SignatureValue>
(<KeyInfo>)?
(<Object>)*
</Signature>
```

where: ? denotes zero or one occurrences,
* denotes zero or more occurrences, and
+ denotes one or more occurrences.

2.3.1 The Reference Processing Model

The signed content, which may be contained in the same document as the XMLdsig and/or external to the document containing the XMLdsig, is referenced with a `<Reference>` element. The URI (Uniform Resource Identifier) [4] 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 [12] 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 XMLdsig's Reference Processing Model [2, Sect.4.3.3.2] specifies that the dereferenced content is supplied to the first transform. As illustrated in Fig. 2, 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 XMLdsig Reference Processing Model is also used for XMLdsig Reference Validation [2, Sect.3.2.1], which is a required part of XMLdsig Core Validation.

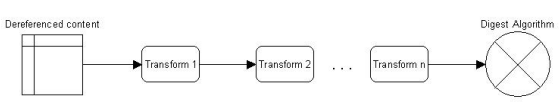

Figure 2: Transform chain for processing content prior to input to digest algorithm.

Adapted from [15, p.720]

2.4 Related Work

Rivest [18] proposed a general concept which has since been referred to as "...signature schemes that admit forgery of signatures derived by some specific operation on previous signatures but resist other forgeries." [3]. In this area, Micali and Rivest introduced transitive signature schemes [16], while Bellare and Neven presented schemes that provided performance improvements [3]. Johnson, Molnar, Song and Wagner have investigated a general approach to homomorphic signature schemes for some binary operations such as: addition, subtraction, union, intersection [14]. Whilst their additive scheme was proved to be insecure, their redactive scheme construction was shown to be secure.

"Digital Credentials" that are issued by trusted "Credential Authorities" have been proposed by Brands and have the capability for selective disclosure of the data fields in the credential [8, 9]. Brands suggests that Digital Credentials can also be used for such things as birth certificates, as well as gift certificates, tokens, train tickets etc., in which case they have appeal as an alternative to identity certificates. Whilst it is envisaged that Content Extraction Signatures could be used for these types of application, they do not require the signing of the content to be performed by a Certification Authority.

A different approach was taken by Orman [17] who proposes an authorisation and validation model, using the XMLsig standard, for mildly active content in web pages. This model enables the content owner's intentions with respect to document modification to be embodied in the document so that any party can validate the document with respect to these intentions. The nature of this work differs from ours in that it enables document composition by more than one author. It permits the document owner to delegate modification rights verifiably by signing the URL: a form of "late binding". Other work has focussed on certain types of path and selection queries over XML documents that can be run on untrusted servers. Devanbu, Gertz, Kwong et al. [11] have proposed a new approach to signing XML documents so that answers to arbitrary queries can be certified.

3. CES-XML SIGNATURE DESIGN

In order to implement our custom transform, which we discuss in § 3.1, we need to organise the XMLdsig structure to accommodate the fragment information. The fragment information is included in `<Object>` elements as shown in Fig. 3, which depicts the overall structure of the XML Signature. The `<Reference>` elements each refer to a corresponding `<Object>` element, which in turn contains the information for each document fragment. Each `<Reference>` element includes a `<Transform>` element specifying our custom transform as follows:[1]

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

Each `<Object>` element contains a `<Fragment>` element that contains either a `<Salt>` or a `<Digest>` element as follows:

```
<Object ID>
  <Fragment URI CEAS>
    [<Salt> | <Digest>]
  </Fragment>
</Object>
```

[1] Prefixes such as http://ace.edu.au/tranforms/ have been omitted throughout for presentation and security reasons.

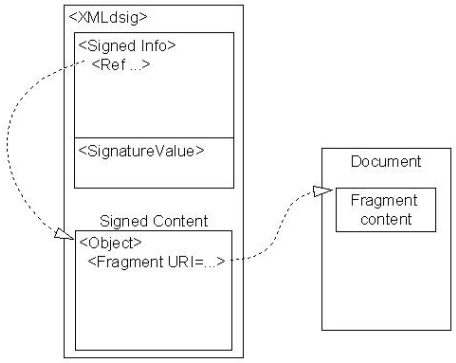

Figure 3: CES-XML signature structure.

Adapted from [15, p.702]

where: | denotes an exclusive OR.

The URI attribute of the fragment references the fragment content while the CEAS (Content Extraction Access Structure) attribute 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 [20]. 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 the document owner, Bob, produces a subdocument, an extracted signature corresponding to the subdocument must be generated so that it can be validated by the subdocument recipient (or verifier), Carol, as being signed by Ace University. This extracted signature has the `<Salt>` element replaced with a `<Digest>` element for the corresponding fragments which are not included (blinded) in the subdocument. The digest value is generated from the fragment content with the salt appended. Therefore, the extracted signature, which is generated for the subdocument, has a `<Salt>` or `<Digest>` element for fragments that are present or blinded respectively.

3.1 The Custom Policy Verification Transform

The requirement for the custom transform is to process the fragment's dereferenced content, check for compliance with the extraction policy and emit a byte stream.

As illustrated in Fig. 4, the transform first checks for the presence of a `<Digest>` element, which indicates that 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 and emitted as a byte stream. Should verification of the extraction policy fail, then two bytes of zeroes will be emitted in place of the digest value. This will ultimately cause reference validation failure and in turn core validation failure as the emitted bytes will not match those originally used to create the digest value stored in the reference's `<DigestValue>` element.

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 then emitted as a byte stream.

In addition to the explicit requirements of the transform, 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.

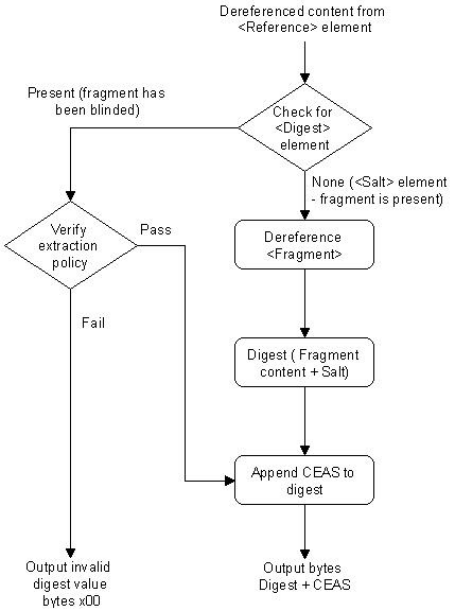

Figure 4: *VerifyPolicy* transform algorithm.

3.2 Revocation Requirement

Discussions with Ace University have revealed an additional requirement: the ability to revoke an academic transcript. The university may need to revoke some academic transcripts due to some problems, or revoke ageing transcripts for re-signing due to the changing of its signing keys. Whilst it is not possible to recall signed transcripts, which have been distributed, it is possible to stop transcripts from being verified, thus causing students to seek the re-signing or re-issue of their academic transcripts.

Building on our earlier transform implementation in §3.1, we propose another custom transform, *CheckStatus*, to assist with the revocation requirement. This can occur while respecting the privacy and anonymity of the student whose academic transcript, or a subdocument thereof, is being validated by a prospective employer.

3.3 Revocation Mechanism

The *CheckStatus* transform could be inserted in the transform chain prior to the *VerifyPolicy* transform, however, this would mean that the status would be checked for every fragment, which is unnecessary and inefficient in this particular case. A better approach is to create an `<Object>` element to contain a `<Transcript-Details>` element with attributes for TranscriptId, CohortId[2], and URI for the remote service. This `<Object>` element is then referenced by the first `<Reference>` element, which specifies the *CheckStatus* transform as follows:

```
...
  <Reference URI="#obj1">
    <Transforms>
      <Transform
```

[2]We use cohort to define a grouping of students which is sufficiently large to offer a good degree of anonymity and yet not so large that it is unwieldy to use. In this case, the university could use a grouping for the students who graduate in each semester, viz. 2002s2—for those who graduate in semester 2, 2002.

```
        Algorithm="ces#CheckStatus" />

...
<Object Id="obj1">
  <TranscriptDetails
        URI="requestStatus"
        TranscriptId="123456"
        CohortId="2002s2" />
</Object>
```

The custom transform, *CheckStatus*, as illustrated in Fig. 5, appends the CohortId as a name-value pair to the URI and sends it to the university's web server as follows: `http://ace.edu.au/transcripts/requestStatus?CohortId=2002s2`

Responding to the request, the university sends either a compressed Revoked Transcripts List (RTL), which is comprised of revoked TranscriptIds, or the CohortId identity if the entire cohort has been expired/revoked. The *CheckStatus* transform then checks the returned list to ensure that the student's transcript has not been revoked. If the transcript has not been revoked, the CohortId, TranscriptId and URI are concatenated and emitted as a byte stream. Conversely, for a revoked transcript, two bytes of zeroes will be emitted, as in §3.1.

A simpler approach would be to send the TranscriptId to the web server and have a boolean type response returned indicating whether the transcript is valid or not. However, this does not respect the privacy of the student as it would leave a trail of the student's academic transcript usage. Instead, sending a CohortId and returning a RTL affords a degree of anonymity to the student as no individual student is identified and the university has no way of identifying the student or whether the transcript being verified will pass or fail.

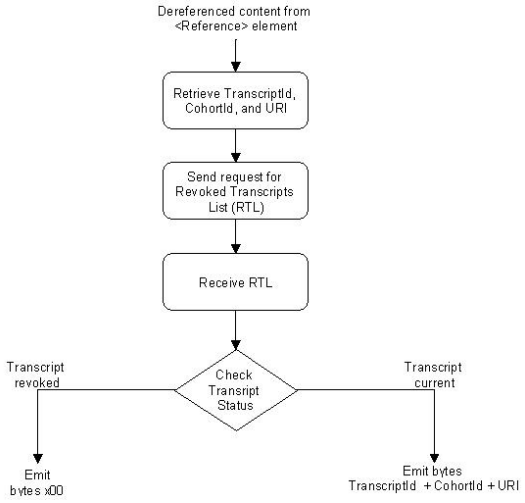

Figure 5: *CheckStatus* transform algorithm.

4. IMPLEMENTATION

Our implementation was performed using the Microsoft .Net Framework XML Signature API and the C# language. A simple, single dimensional signer's extraction policy will be used in the implementation where, for each fragment, the CEAS will be used simply to specify whether the fragment may be blinded. Therefore, we can assign each fragment's CEAS attribute a value of 1 if it is allowed to be blinded, or 0 otherwise.

4.1 Transforms On-Demand

Our implementation approach to add the custom CES extensions to the standard XML signature processing involved dynamic invocation of a custom transform that is used in the signing and verifying operations.

Dynamically retrieving the custom transform by the verifier immediately raises security concerns. Since the download will occur across the Internet, which is typically insecure, the code may be subject to server redirection through DNS spoofing [19] and proxy server interceptions, as well as modification both on the server as well as in transit. Through support of the Secure Socket Layer (SSL) protocol in the .Net Framework we are able to securely download an assembly (a unit of code) directly from the source document's referenced Web site [13; 15, pp. 229–38]. This can be specified by the author when signing the document by using an HTTPS URI [13] for the transform. To combat the risk of the code being modified or replaced with malicious code either on the server or through server redirection, we can use a digital signature to sign the code. Since the code is digitally signed it is not possible to modify the code without detection. This is also supported in the .Net Framework [15, pp. 126-30]. In doing this, we greatly increase the level of code security and trust by overcoming the insecure communication channels in addition to proxy and DNS vulnerabilities in much the same way as major software house use for their own code packages for download.

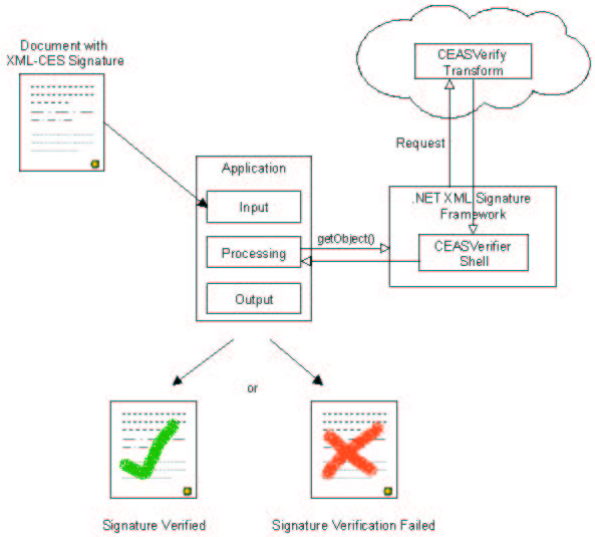

Figure 6: Overview of framework for dynamic downloading of CEASVerify algorithm

As depicted in Fig. 6 the signed *CEASVerify Transform* is downloaded by the signed *CEASVerifier Shell* within the .Net XML Signature Framework. The framework returns the local *Transform* object to the calling application that invokes the available methods which map into the *CEASVerify Transform* object downloaded from the remote server.

Our implementation relies on the embedding of a URI to point to the signer's web site which stores the transform code to be downloaded. This code matches the document to be verified so that the custom verification matches the custom signing. Since a signed document contains the URI as a part of the signature, it allows the document author to control integrity of the algorithm and the document.

By relying on versioning information provided in the .Net Frame-

work, our solution provides an elegant model for on-demand code downloading to perform an operation that is integral to the signing and verifying operations. This also future-proofs our CES model, which may undergo further internal optimizations. Since we have this level of control, current documents in circulation will continue to be valid, while new documents will refer to their new respective verifier code bases.

We view this as a highly flexible approach that is transparent to the user and does not break once new versions of CES documents and algorithms are released. Our only restriction is that this is tied to the Microsoft .Net Framework, but we envision that a parallel implementation is entirely feasible using the Java environment or other Remote Procedure Call systems.

Furthermore, we anticipate the usage of XML Web Services as another means of code distribution and signer/verifier binding. Since SOAP (Simple Object Access Protocol) [5] permits remote method invocations we are also investigating how our model may benefit from its usage. However, since SOAP security remains incomplete, mainly in encryption and message signing, we anticipate that our current implementation will not be moved to a SOAP implementation until the current work in Web Service Security matures and becomes widely adopted [1].

4.2 Non-XML Content

The XMLdsig specification is designed to sign both XML and non-XML content. Raw data or binary files, such as images etc., are handled through the use of the Base64 transform [2, Sect. 6.6.2]. However, there appears to be no direct support for signing parts, or fragments, of non-XML content or unstructured files. For example, signing all of the individual classes, or packages, to be found in a software source distribution, such as the Redhat Linux distribution. Our implementation, on the other hand, involves signing fragments in a plain text file and storing the CES in an associated XML file. This type of XMLdsig is called a *detached signature*.

To address the fragment content in the text file we simply used fragment offset and length parameters. The *VerifyPolicy* transform parses these parameters from the fragment component of the URI as per [4]. The fragment identifier of the URI attribute is used to specify the fragment content parameters as follows:

```
...
<Object Id="obj">
  <Fragment
    URI="acad-transcript.txt#offset.length"
    ...>
</Object>
```

4.3 Discussion

Our implementation provided for a single dimension extraction policy where fragment grouping as in [10] was not initially considered. The policy checking mechanism uses the Reference Processing model and is inserted into the `<Reference>` element's transform chain. Using this approach has the limitation that as the transform chain is executed it proceeds within a scope that is relative to the current `<Reference>` element being processed. The problem that arises is that to handle fragment grouping, the *VerifyPolicy* transform needs to be able to access other `<Reference>` element contents, which are effectively out of scope. In view of this limitation, and reflecting on our original design for the CES-XMLdsig structure, it seems that configuring a `<Reference>` element for each `<Fragment>` element is not very efficient, nor is it necessary. An alternate, more efficient, approach has emerged that also accommodates the handling of fragment extraction policies that include grouping of fragments. This will also require a re-design of the *VerifyPolicy* transform to handle this capability and is included in on-going work to investigate a more efficient and flexible alternative.

A seemingly obvious and tempting approach is to simply add an earlier `<Reference>` element to include signing the custom transform along with the document in an effort to ensure integrity prior to using it. This will also bind the algorithm to the document. However, the problem with this approach is that the custom transform will still be executed as a transform in the later `<Reference>` elements that use it. If the custom transform has been tampered with, it will cause *Reference Validation* failure and in turn *Core Validation* failure. The XMLdsig processing model does not appear to allow for aborting the verification process part way through. The model expects to complete processing and relies on a result comparison to determine the verification outcome. Instead, an approach must be taken that allows execution of the downloaded code *only after* the code has been verified.

Our approach to signing and verifying a document using dynamically loaded transforms further blurs the notion of what is being signed. We are not only signing the document content, but we are also signing functionality with respect to the content. Using the code signing support provided by the .Net Framework to protect the dynamically loaded custom transform does not bind the transform to the document. Depending on the requirements, the signer may not want to be constrained by having the algorithm bound to the document in an orthodox approach. Rather, the signer may prefer to reserve the ability to update the custom transform with newer signed versions.

5. PRACTICAL USE OF CES FOR ACE UNIVERSITY

Utilising the custom transforms we now provide a brief description of the use of CES in our example from § 2.1 from the perspective of the following roles: the signer, document owner, and the verifier.

5.1 Signing

The university defines the fragments and the extraction policy for Bob's academic transcript and proceeds to sign the transcript using a CES. Input to the signing operation includes Bob's academic transcript, the university's private key, TranscriptId, CohortId, and the *CheckStatus* and *CheckPolicy* transforms. Upon completion, the signed transcript is forwarded to Bob.

5.2 Extracting

The document owner, Bob, proceeds with blinding his DOB to generate an academic transcript subdocument. On completion, the new subdocument is written to disk along with a new constructed matching CES. It is not necessary for Bob to go back to the university to get it to sign his academic transcript extract. Nor is it necessary for Bob to be online for this process as the blinding and signature generation operation does not require the use of either of the transforms. The custom transforms are only necessary for signing and verification of the document. Therefore, if Bob is prepared to proceed without first verifying the CES on the academic transcript, he can blind permissible content in an offline mode. However, he will also not be able to verify the newly created academic transcript subdocument whilst offline.

5.3 Verifying

The prospective employer, Carol, must be online in order to verify Bob's academic transcript extract with its CES. Verifying the university's CES involves checking the following, which have now been built into the XMLdsig format:

- the transcript is current, i.e. it has not been expired/revoked,
- extracted content complies with signer's extraction policy.

Through the implementation of the custom transforms, these requirements have been included in the XMLdsig Reference Validation process. For Carol to verify Bob's academic transcript she requires the university's public key and the transforms: *CheckStatus* and *VerifyPolicy*.

As illustrated in Fig. 7, the verification application retrieves the *CheckStatus* transform from an external source. The transform is then invoked whereby it retrieves a RTL by sending the CohortId as a request. Next, the *VerifyPolicy* transform is retrieved and invoked whereupon it checks the fragment content against the extraction policy. Although a document may have many fragments, there is only a single *VerifyPolicy* transform request instead of multiple requests, since the transform is cached for further use.

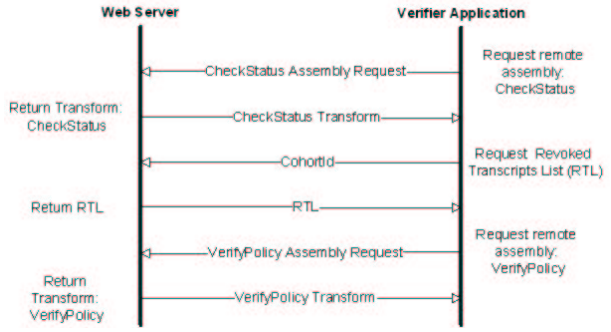

Figure 7: Sequence diagram for external algorithm and information exchange for the Verifier Application in the University System

6. CONCLUSIONS

The authors of the XML Digital Signature specification [2] provided flexibility for the use of custom transforms beyond the ones defined in the specification. In this paper we have shown our design for *CES functionality* using the XMLdsig specification. This was achieved with the help of a custom transform algorithm. We also dealt with an additional requirement to revoke an academic transcript by further exploiting our custom transcript approach. We discussed our implementation for non-XML content and described how we achieved dynamic loading of the transforms for signing and verifying documents.

Our implementation using the XMLdsig specification provides interoperability for the verifier in an open system. Utilising custom transforms to achieve *CES functionality* and revocation of the verification mechanism demonstrates that custom signing and verification solutions can be developed as required using XMLdsig. The use of custom signing and verifying solutions is not restricted to a 'closed system'. Through the use of dynamic loading we have shown that the verifier can still verify a custom XMLdsig compliant signature even though a custom signature was produced.

By developing and implementing our own custom transforms to use according to the XMLdsig Reference Processing Model we can utilise the flexibility provided by the XMLdsig specification to expand the functionality of XMLdsigs to include CES functionality.

Further work to enable the handling of richer more complex extraction policies involving grouping is being investigated.

7. REFERENCES

[1] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon. Web services security (WS-Security). In C. Kaler, editor, *Version 1.0*. Apr. 05 2002. [Last accessed: February 24, 2003] `http://www-106.ibm.com/developerworks/library/ws-secure/`.

[2] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML-signature syntax and processing. In D. Eastlake, J. Reagle, and D. Solo, editors, *W3C Recommendation*. Feb. 12 2002. [Last accessed: September 18, 2002] `http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/`.

[3] M. Bellare and G. Neven. Transitive signatures based on factoring and RSA. In Y. Zheng, editor, *Proceedings of The 8th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2002)*, volume 2501 of *Lecture Notes in Computer Science*, pages 397–414. Springer, December 2003.

[4] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 2396. uniform resource identifiers (URI): Generic syntax. Available online, August 1998. [Last accessed: September 25, 2002] `http://www.ietf.org/rfc/rfc2396.txt`.

[5] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. *Simple Object Access Protocol (SOAP) 1.1*. W3C note 8 May, 2002 edition, 2002. [Last accessed: November 15, 2002] `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`.

[6] J. Boyer, D. Eastlake, and J. Reagle. *Exclusive XML Canonicalization*. W3C Recommendation 18 July, 2002 edition, 2002. [Last accessed: July 12, 2002] `http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/`.

[7] J. Boyer, M. Hughes, and J. Reagle. *XML-Signature XPath Filter 2.0*. W3C Recommendation 08 November, 2002 edition, 2002. [Last accessed: November 12, 2002] `http://www.w3.org/TR/2002/REC-xmldsig-filter2-20021108/`.

[8] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, 2000.

[9] S. Brands. A technical overview of digital credentials. Available online, Feb. 20 2002. [Last accessed: February 18, 2003] `http://www.xs4all.nl/~brands/overview.pdf`.

[10] L. Bull, J. Newmarch, and Y. Zheng. Enhancing privacy through selective disclosure of verifiable content or the facts, the whole facts, and nothing but the facts. Technical Report 2002/123, School of Computer Science and Software Engineering, Monash University, 900 Dandenong Road, Caulfield East, Victoria 3145 Australia, October 2002.

[11] P. T. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. G. Stubblebine. Flexible authentication of XML documents. In *ACM Conference on Computer and*

Communications Security, pages 136–45, 2001.

[12] N. Freed and N. Borenstein. Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies. Available online, 1996. [Last accessed: October 16, 2002] `http://www.ietf.org/rfc/rfc2045.txt`.

[13] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0. Available online, 1996. [Last accessed: September 18, 2002] `http://wp.netscape.com/eng/ssl3/draft302.txt`.

[14] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In *Proceedings of the RSA Security Conference Cryptographers Track*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–62. Springer, February 2002.

[15] B. LaMacchia, S. Lange, M. Lyons, R. Martin, and K. Price. *.NET Framework Security*. Addison-Wesley, Boston, MA, 2002.

[16] S. Micali and R. L. Rivest. Transitive signature schemes. In B. Preneel, editor, *Proceedings of The Cryptographer's Track at the RSA Conference (CT-RSA 2002)*, volume 2271 of *Lecture Notes in Computer Science*, pages 236–243. Springer, December 2002.

[17] H. Orman. Data integrity for mildly active content. In *Proceedings of Third Annual International Workshop on Active Middleware Services*, pages 73–7. IEEE Computer Society, March 2002.

[18] R. Rivest. Two signature schemes. Available online, October 2000. Slides from talk given at Cambridge University. [Last accessed: February 19, 2003] `http://theory.lcs.mit.edu/~rivest/publications.html`.

[19] D. Sax. DNS spoofing (malicious cache poisoning). Available online, Nov. 12 2000. [Last accessed: February 25, 2003] `http://www.sans.org/rr/firewall/DNS_spoof.php`.

[20] R. Steinfeld, L. Bull, and Y. Zheng. Content extraction signatures. In *Proceedings of The 4th International Conference on Information Security and Cryptology (ICISC 2001)*, volume 2288 of *Lecture Notes in Computer Science*, pages 285–304. Springer, December 2001.