Secure Web Forms with Client-Side Signatures

Mikko Honkala and Petri Vuorimaa

Telecommunications Software and Multimedia Laboratory,
Helsinki University of Technology,
P.O. Box 5400 FIN-02015 HUT FINLAND
{Mikko.Honkala,Petri.Vuorimaa}@tml.hut.fi

Abstract. The World Wide Web is evolving from a platform for information access into a platform for interactive services. The interaction of the services is provided by forms. Some of these services, such as banking and e-commerce, require secure, non-repudiable transactions. This paper presents a novel scheme for extending the current Web forms language, XForms, with secure client-side digital signatures, using the XML Signatures language. The requirements for the scheme are derived from representative use cases. A key requirement, also for legal validity of the signature, is the reconstruction of the signed form, when validating the signature. All the resources, referenced by the form, including client-side default stylesheets, have to be included within the signature. Finally, this paper presents, as a proof of concept, an implementation of the scheme and a related use case. Both are included in an open-source XML browser, X-Smiles.

1 Introduction

Commerce and communication tasks, such as ordering items from a shop or using e-mail are becoming popular in the World Wide Web (WWW), and therefore WWW is transforming from a platform for information access into a platform for interactive services [1]. Unfortunately, some of the technologies used today are outdated and, infact, were not originally designed for the complex use case scenarios of today's applications, for instance, secure transactions.

The Wold Wide Web Consortium (W3C) has recently developed a successor to HyperText Markup Language (HTML) forms, called XForms [2]. It removes need for scripts from the forms and separates the form's model from the presentation. It can use any XML grammar to describe the content of the form.

Digital signatures are a primary way of creating legally binding transactions in information networks, such as WWW. Unfortunately, digital signatures are hard to apply to HTML [3]. Using eXtensible Markup Language (XML) helps, since a conforming XML processor is not allowed to accept non-well-formed XML. Also, W3C and Internet Engineering Task Force (IETF) have specified a standard way of signing XML data, i.e., XML Signature.

XML Signatures is a language, which provides the necessary framework for encoding, serializing, and transmitting signatures in XML format. The main focus is signing XML, but it can also function as the signature description language

for any ASCII or binary data. In that case, the additional data is included in the signature as a reference. [4].

Why are client-side signatures needed? In order to be legally binding, the signature must be created in a secure way using the private key of the signer. Usually, this can be done in a tamper-proof smart card or a similar device. Because of this requirement, it is not possible to create the signature at the server side along with the other application logic.

Why to integrate signatures into XForms? Signature is usually attached in data produced by the user, not by the author, and XForms is a technology for collecting user input. Signature must be serialized and transmitted to the receiver. XForms submission is a good way of submitting digital signatures encoded in XML.

How to make signatures over forms legally binding? Signatures can be made legally binding in many countries, if the signing process fulfills some requirements. A basic requirement, apart from the technical requirements for the signature, is that the signer must see everything that she is signing. Also, it must be possible to reconstruct the document that the signer has signed later.

1.1 What You See Is What You Sign

An important, but often overlooked, property of a signing application is the capability to express the signature over everything that was represented to the user. This principle is usually called "What you see is what you sign" (WYSIWYS). To accomplish this, it is normally necessary to secure as exactly as practical the information that was presented to that user [3]. This can be done by literally signing what was presented, such as the screen images shown to a user. However, this may result in data, which is difficult for subsequent software to manipulate. Instead, one can sign the data along with whatever filters, style sheets, client profile or other information that affects its presentation. [4]

1.2 Related Work

Previous work in the field includes, e.g., Extensible Forms Description Language (XFDL) [3]. In XFDL, all information related to the form is included in a single XML file, including form definition, styling information, form data, and even binary attachments, and the signature is created over this single file. This approach does not fit well to signing Web pages consisting of decoupled resources, such as stylesheets and images, which is the target for this paper.

There is also research on specific algorithms on determining whether unsigned areas are visually overlapping with signed areas [5]. This is required, if signatures over a partial form are allowed. Unfortunately, this approach requires certain type of layout, and thus it cannot easily be extended to languages with different layout models, such as SVG (absolute layout) and XHTML (flow and absolute layout), or different modalities, such as Voice XML (speech modality).

2 Research Problem

The research problem of this paper is *how to create legally binding secure services in the WWW*. Secure transmission technologies, such as Secure Sockets Layer (SSL), are already widely used, but they do not support the notion of a client-side signature. Thus, the main focus is to enable the services on the WWW to allow digital signatures over the user's input. This is achieved by allowing XForms forms to be signed by the user. A subproblem is to ensure that the user has a clear understanding what she is signing (WYSIWYS). Another important subproblem is to ensure the full reconstruction of the signed form, when validating the signature.

2.1 Use Cases

We have identified three basic use cases that should be supported by the scheme:

USE CASE 1, Single form: In the most basic use case, the user downloads a form, which is to be signed. She fills the form and initiates the signing process. Within the signing process there is the possibility to select the key, which is used in the signing. When the form is signed, she submits the form. The form can, for instance, be an email that she fills and signs before sending.

USE CASE 2, Form approving: When the first user has signed the form, the form is sent to her supervisor, who adds some data, and then signs it with her own key. The supervisor's signature should also cover the already signed portion of the form, since she approves also the data signed by the first user.

USE CASE 3, Multiparty form: Third use case, a joint insurance claim filing, is depicted in Fig. 1. Multiple parties are filing a single insurance claim. It should be possible to add new parties and attachments, but each of the parties signature must not allow changes in the core information of the claim form.

In all use cases 1-3, it must be possible to verify the resulting signature's integrity at any future time, and any change to the signed form, including all referenced items (e.g., stylesheets, images, etc.), must invalidate the signature.

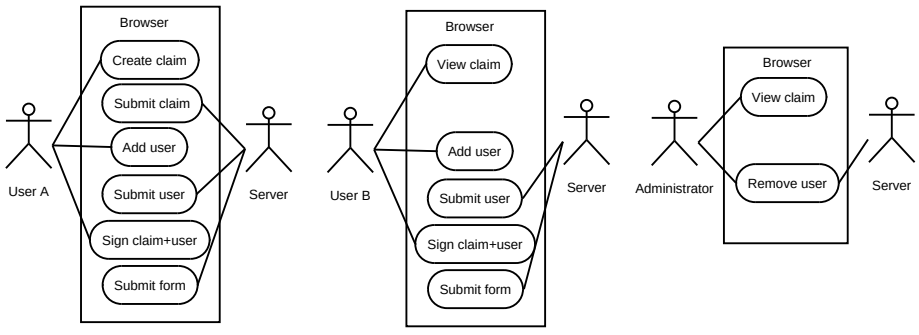

Fig. 1. Use case 3: Multiparty form.

2.2 Requirements

From the basic use cases, the following requirements for the XML Signature and XForms integration were gathered:

Signature security *Technical requirements for secure signatures.*
- **Client-side** The signature must be generated client-side so that the user can check the signature validity before submitting. Also, support for signing with secure smart card must be supported.
- **Common algorithms** The signature must be generated using common, trusted, algorithms for maximum security.
- **Signed form reconstruction** It must be possible to reconstruct the signed form in case of dispute.

Signature coverage *Which parts of the form need to be signed (WYSIWYS).*
- **User input** The data user inputted through the form.
- **UI** The UI document, which describes the layout of the form.
- **All referenced data** Stylesheets, images, objects, applets, scripts, schemas, external instances, etc.
- **The user agent info** Information about the user agent.

Complex signature support *Support for complex signing scenarios.*
- **Partial signature** Support signing only part of the form.
- **Multiple signatures** Support multiple signatures within one form.

Form language integration
- **Ease of authoring** Provide as easy syntax as possible for authors.
- **Ease of implementation** Use of off-the-shelf libraries should be possible.
- **Modality and host language independence** The design should be independent of modality and host language.

An XForms form is independent of its presentation, which fulfills the **Modality and host language independence** requirement. As a language, XForms requires a host language in order to realize the presentation. It must be kept in mind, that the layout strategy (e.g., SVG, XHTML) or even the modality (e.g., Voice XML) is not fixed when using XForms. That is why the goals of this paper differ, e.g., from [5], which expects a box layout, and based on that is able to have fine-grained author control over the signature.

3 Design

We considered two different approaches to the XForms XML Signature integration, keeping in mind the requirements above. First option was the addition of new submission filter, which would create the signature at submission time. The second option, which was chosen, was to create a new XForms compatible action for the signature processing. This is more flexible than the submission filter, and it fits better to the XForms processing model, thus fulfilling the **Form language integration** requirement. The only drawback is that the author must explicitly have a placeholder in the XForms instance data, where to store the signature.

Since XForms can represent dependencies between fields in the form, it is hard to implement the **Partial signature** requirement at the client side, while fulfilling the WYSIWYS paradigm. Because of this, in our scheme, the whole form is always signed with the related resources, and no signatures over a partial form are allowed. Thus, the **Partial signature** requirement must be fulfilled at the server-side. Fortunately, XForms provides functionality to make this task easier, and we demonstrate this with an implemented use case in Sect. 5.1.

3.1 Process Description

The overall process in the scheme can can be divided into three steps: 1) Retrieving and filling the form, 2) attaching the digital signature into the form, and submitting the signed form into the server, and 3) validating the submitted form data and the signature. This is depicted in Fig. 2.

There are four main actors involved in the process: a) *the user* fills and signs the form, b) *the browser* displays the form, and creates the signature, c) *the smart card* signs the hash of the signed form, and d) *the server* validates the submitted form data and the signature. If a lesser security level is appropriate, the hash can be signed using the browser instead of the smart card.

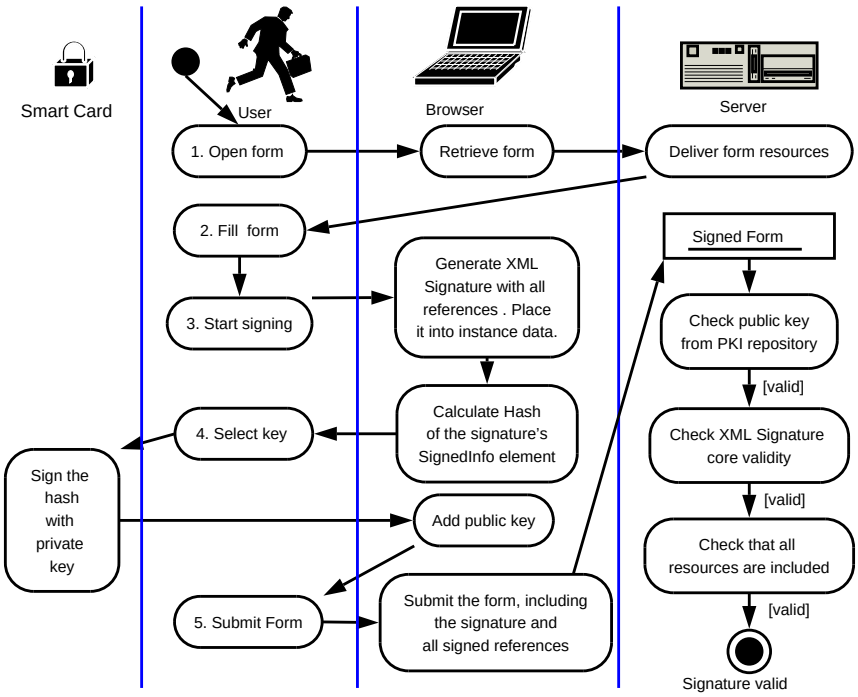

Fig. 2. The overall process of creating and validating signatures.

3.2 Signature Creation With XForms Extension "sign"

XForms language was extended with a namespace *"http://www.xsmiles.org/2002/signature"* and a single element *sign* in that namespace. The element integrates to XForms processing model in two ways: first, it is an XML Events action, similar to *xforms:send* [2]. Second, it uses the XForms single node binding to specify where to store the signature.

Attribute: to an XPath expression specifying the node, under which the signed content is placed. All previous children of that node are destroyed. The result of the evaluation must be a document or a element node. If the node is the document node, then the content is placed as the document element.

Operation is as follows:
The signature operation consists of the following steps, each of which must follow XML Signatures' *core generation* rules [4]. See Fig. 2 for overall description of the process.

1. Disable any user stylesheets. These stylesheets might prevent some parts of the form from displaying.
2. If the document is not a top-level document (e.g., is inside a frameset or embedded in another document), abort signature creation. This ensures that the user has clear view of the complete content he will be signing.
3. Evaluate the XPath expression in *@to* attribute, using the XForms' default context node (first model, first instance, document element) as the context node and context nodeset, and the sign element as the namespace context element, and XForms functions in scope. The node *"target"* is the first node in the resulting nodeset. If the result is not a nodeset, or the first node is not an element or document node, the processing stops.
4. Remove all child nodes of *"target"* node.
5. Create a signature with the following references:
   - Always create an enveloping signature.
   - Create an <dsig:Object id="x"> and <dsig:Reference URI="#x"> elements for each live instance data, and copy to contents of the live instance inside the object element. The id 'x' must be replaced by an arbitrary unique id.
   - Create <dsig:Object id="y"> and <dsig:Reference URI="#y"> elements for all user agent and user stylesheets, which have been used for displaying the form, and copy the stylesheets as the text content of the root node of the object. The id 'y' must be replaced by an arbitrary unique id.
6. Create detached references to all URLs referenced by the host document:
   - The host document
   - All referenced URLs separately: images, objects, applets, stylesheets, scripts, XForms external instances, xinclude, xlink, XSLT, etc.
7. Create a valid signature over all the references with the users private key.
8. Create a valid dsig:KeyInfo element containing the signers public key.

9. Place the dsig:Signature element as the only child of the *"target"* node.

Example of the usage of the signature action is below. The XForms trigger element contains the sign element, which creates an enveloping signature to an empty instance with id "signature", when the DOMActivate event is caught, for instance, when the user activates the trigger.

```
<trigger>
  <label>Sign message</label>
  <sign:sign to="instance('signature')/.." ev:event="DOMActivate"/>
</trigger>
```

Note that all resources that are used to render the document must be included in the signature. This includes images, objects, applets, stylesheets, scripts, XForms external instances, xinclude, xlink, XSLT, etc. The resources that are fetched using an get operation with an URL, are added as detached references.

The user agent has its own default CSS stylesheets. The signature must contain all stylesheets that have been applied when the page has been displayed. Since the default stylesheets cannot be identified with an URL, they must be included as inline objects within the signature.

3.3 Event: "signature-done"

In order to notify the form after the signature has finished, a notification DOM event *signature-done* was added. It's target is the sign action element. It bubbles, is cancellable, and does not have any context information. The signature-done event does not have any default action. It allows the creation of XML Events listeners for a certain sign action element.

3.4 Signature Validation

The validation of the signature has the following steps (cf. Fig. 2 for a general description of the process):

1. Find the Signature element from the submitted instance data.
2. Read the public key from the KeyInfo element and check from a Key repository that it corresponds the users identity. If it does not, abort the validation process.
3. Validate the Signature element according to the XML Signatures' core validation rules [4]. If the validation fails, abort the validation process.
4. Do application-specific validation of all resources. For detached references, this is simple: just check that the URL is correct (the hash and the signature has been checked in the previous step already). For enveloped resources, application-specific logic must be included in the validation. For instance, XForms calculations, defined in the form, should be run in the server-side, in order to check the correctness of the submitted instance data. If any check fails, abort the validation process.
5. The Signature is accepted if none of the above checks fails.

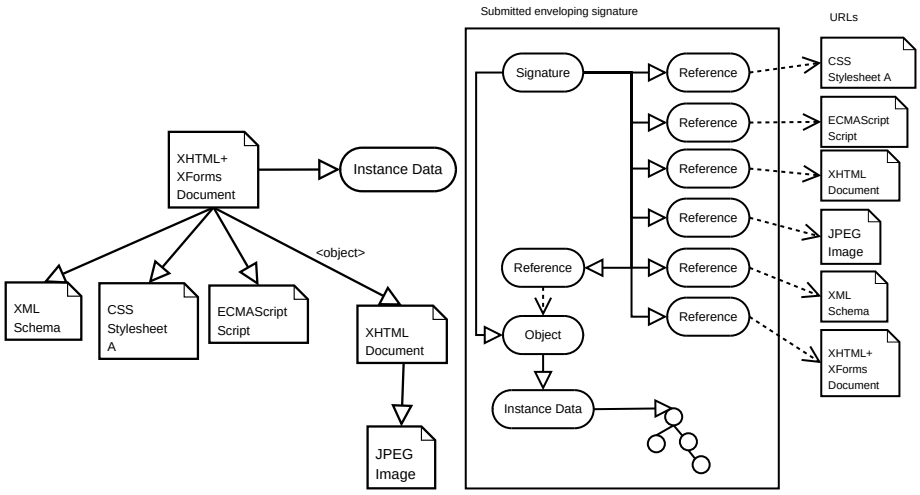

Fig. 3. Left: A Document with references. Right: The signature over the document.

4 Implementation

The first author has implemented the current complete implementation with the processing described in this paper. It is based on an earlier experimental implementation, which was done in co-operation with Heng Guo [6].

4.1 X-Smiles XML Browser

The signature processor was integrated with the XForms processor in the X-Smiles browser [7]. X-Smiles is a modular XML browser, which allows registering different components for different XML namespaces, thus allowing easy creation of XML "plugins", which are called Markup Language Functional Components (MLFC) [8]. The sign element and it's namespace were created as a new MLFC. Note that the signature MLFC is purely a processing component, i.e., it does not have an associated UI rendering component.

4.2 Implementation Cost When Integrating to the XForms Processor

A requirement for building modular software is to decouple components and use well-defined interfaces for inter-component communication. In the integration of XML Signature and XForms processors, we used the DOM interfaces as much as possible. The signature element is part of the UI DOM tree, so it can naturally access the tree with the XML and XForms DOM access. A DOM 3 XPath component for XPath evaluation, extended with XForms functions, is used by the @to attribute. We have decided that the XForms context node inheritance rules do not apply to the sign element, since it is not in the XForms namespace.

The XML signature is created using the *Apache XML Security 1.1 for Java* library[1]. This library is a complete XML Signature implementation. It is also responsible for creation of the external references in the signature.

The user agent must provide an interface to access the list of resources, which were loaded for the form being signed. Some of these resources can be referenced via an universal URL (such as the images), while others are local to the processor. Both must be provided through this interface.

5 Results

5.1 Use Case: Joint Insurance Claim

We implemented a joint insurance claim application, which implements the use case *Multiparty form*, using the proposed scheme. The application consists of one XHTML+XForms page and a server-side process (servlet).

In the application, it is the responsibility of the servlet to filter the instance data so that, at each step, the correct parts of the insurance data are transmitted to the client. For instance, since the signatures are not related to each other (remember that it must be possible to add new signers or remove them without affecting the validity of the other signatures), the servlet filters out the other user's data and signatures for the "add user" view. Similarly, in the verify view, only the related users' data is included. Finally, in the "view claim" view, nothing is filtered out, resulting in the complete view of the claim (cf. Fig. 4). The filtering is implemented using simple XPath statements in the servlet.

The use case and it's source code are included in the X-Smiles distribution. The application was created as a Java servlet. The main logic of the servlet was to store the signatures and the form data and to filter the data according to the current view of the user. The servlet's Java files have 1300 lines, half of which are general utility functions, which can be reused in similar applications.

5.2 Memory Requirements

The XML Security library was stripped down in order to facilitate the integration to the X-Smiles browser distribution. The XML encryption features were removed from the library. The final storage size of the library is 331 KB. The storage size of classes in X-Smiles, related to the implementation, is 25 KB. The total storage size is then 356 KB, which is small enough to be added to the binary distribution of X-Smiles, which is about 8 000 KB of size.

5.3 Applicability to HTML Forms

The presented scheme works for XForms forms. XForms language has certain attributes, which make applying digital signatures easier. The main attributes

[1] XML Security for Java and C++, Software, Apache Foundation. http://xml.apache.org/security/

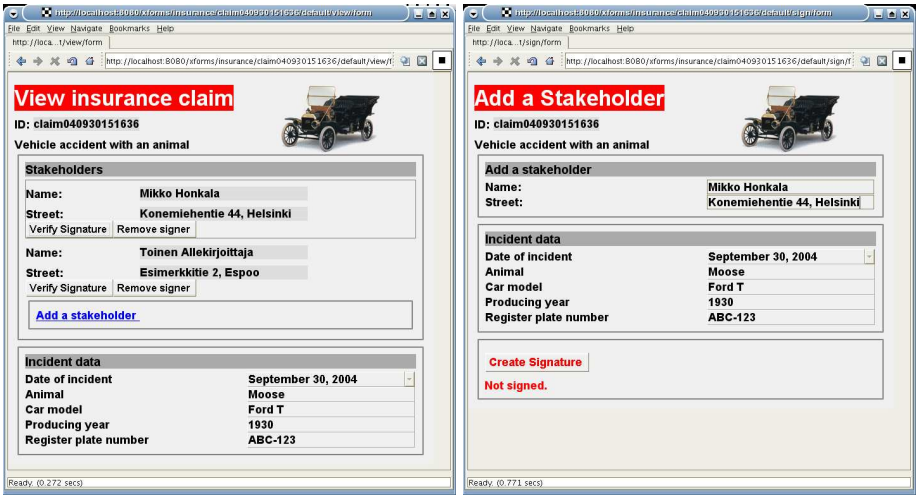

Fig. 4. Viewing the claim and attaching new signers.

are the lack of scripting and a pure XML format. Also, the state of the form is reflected exclusively in the instance data, and not in the presentation DOM.

The scheme could be extended to cover XHTML or even HTML forms as well. XHTML forms without scripting is the easiest case; the outgoing message (e.g., the URL encoded form submission), needs to be signed as a enveloped binary reference. The XHTML document can be signed as an detached reference, either as XML or binary.

Scripting makes the case a bit more complex, since the state of the form is captured in the presentation DOM. The signature in that case, should include the serialized DOM at the moment the signature was created. XHTML serialization is rather straight-forward, but HTML serialization is more complex, foremost because browsers tend to accept and fix ill-formed HTML content, making machine-validation of the serialized HTML difficult, even impossible.

5.4 Complex Signatures

Use case 1, *Single form*, presented in Sect. 2.1, can directly be solved using our proposed extension. We have demonstrated use case 3, *Multiparty form* using a combination of our extension and a server side process, where the server side process has to filter the data sent to the client according to the current view. Use case 2, *Form approving* can be solved in a similar fashion, where the server side process serves the same data in slightly modified form for different signing parties. For instance, the first party is allowed to input the data in the form, and sign it, while for the supervisor, the data and the signature of the first users just shown, and he is only allowed to create a signature on top of it. Validating the signatures in case of a dispute is still straightforward.

5.5 Security Analysis

WWW has an open model, where any compatible user agent can connect and use internet services. This adds extra requirements for the security model of the scheme. First assumption is that the user trusts the user agent, which she is using. That means, that the user agent itself cannot be hostile, and for instance, show the user different content to what it signs. Second assumption is that a key distribution scheme, such as PKI, is used. This way, the server can be sure about the identity of the signer. The signature itself should be done on a smart card or similar device, which does not allow exporting the private key. The scheme presented here detects if a third party inserts content into the form. The signature over this kind of form does not validate, since the server detects a faulty hash for the detached reference.

CSS layout model allows re-flowing the content based on the browser windows width. Also, it allows the use of layers (i.e., absolute and relative blocks), thus potentially hiding content of the form for some browser window widths. Therefore, the author of the form should check the form with all screen widths. Best option is not to use layers in the layout at all. Additional security could be provided by adding a screen dump of the form as an enveloped reference.

6 Conclusion

This paper describes a successful research and implementation of secure Web based services. A extension was added to the XForms language, allowing authors to add digital signing features to XForms applications, without the use of scripting or other client-side languages. The extension was implemented in the X-Smiles XML browser, and a demonstration application was created. The application is included in the online demos of the browser distribution.

After analyzing three use cases and the XForms language, four groups of requirements for the signature integration were gathered: *Signature security*, *Signature coverage*, *Complex signature support*, and *Form language integration*. These requirements lead to design of one declarative action called *sign* and one DOM event called *signature-done*. The action creates a enveloping signature that covers all information related to the form, thus fulfilling the *WYSIWYS* paradigm.

The integration is therefore quite simple at the XML language level. This is good, since it provides ease of authoring. The integration was implemented in a real user agent, X-Smiles, in order to asses the implementation cost of the integration. The results from the integration is that only few hidden properties of the XForms processor need to be exposed, i.e., the integration can be done at the DOM level. The user agent must provide an interface to determine the different resources, which are related to a document that is presented to the user. These resources include images, stylesheets, etc. Some of these resources are never available in the DOM (e.g., the user agent default stylesheet), and therefore they should all be exposed using the same interface.

7 Future Work

The scheme presented in this paper always creates enveloping signatures, and it uses the default signature and canonicalization algorithms defined in the XML Signature specification. While this solves most of the use cases, sometimes it is needed to have better control over the type of signature (e.g., enveloped instead of enveloping) and the signature algorithms. This is left as a future work item.

In the implementation, the user agent information is not included in the signature, which can be a problem when disputes are being solved. A possible future work item would be to include some information about the user agent, which was used to create the signature. One possibility would be to include the Composite Capabilities / Preferences Profile (CC/PP) description of the client as one of the signed objects. CC/PP is not currently very widely adopted, and therefore a simpler scheme could be needed.

Some processing markup languages allow dereferencing external URLs and placing the content of the URL inline within the host document. Examples are XInclude and XHTML 2.0 @src attribute. It is an open question whether to process these languages and place the additional content within the host document or add the referenced URLs as external references in the signature.

Currently, the implementation does not check whether the document, which is to be signed, is a top-level window or not.

Acknowledgment
This research was funded by the TEKES GO-MM project to whose partners and researchers the authors would like to express their gratitude. We would also like to thank Mr. Pablo Cesar for many valuable comments about this paper.

References

1. Hostetter, M., Kranz, D., Seed, C., C. Terman, S.W.: Curl, a gentle slope language for the web. World Wide Web Journal (1997)
2. Dubinko, M., et al. (eds.): XForms 1.0. W3C Recommendation (2003)
3. Blair, B., Boyer, J.: XFDL: creating electronic commerce transaction records using xml. In: WWW '99: Proceeding of the eighth international conference on World Wide Web, Elsevier North-Holland, Inc. (1999) 1611–1622
4. Bartel, M., et.al.: XML-Signature syntax and processing. W3C Recommendation (2002)
5. Boyer, J.M.: Bulletproof business process automation: securing XML forms with document subset signatures. In: Proceedings of the 2003 ACM workshop on XML security, ACM Press (2003) 104–111
6. Guo, H.: Implementation of secure web forms by using XML Signature and XForms. Master's thesis, Helsinki University of Technology (2003)
7. Vuorimaa, P., Ropponen, T., von Knorring, N., Honkala, M.: A java based XML browser for consumer devices. In: 17th ACM Symposium on Applied Computing, Madrid, Spain (2002)
8. Pihkala, K., Honkala, M., Vuorimaa, P.: A browser framework for hybrid XML documents. In: Internet and Multimedia Systems and Applications, IMSA 2002, IMSA (2002)