Document-centric XML workflows with fragment digital signatures

Phillip J. Brooke\(^1\), Richard F. Paige\(^2\) and Christopher Power\(^2\)

\(^1\)School of Computing, Teesside University, Middlesbrough TS1 3BA, U.K.
\(^2\)Department of Computer Science, University of York, York YO10 5DD, U.K.

SUMMARY

The use of digital document management and processing is increasing. Traditional workflows of paper forms are being replaced by electronic workflows of digital documents. These workflows often require multiple signatures to be added to the documents for authorization and/or integrity. We describe examples of digital workflows that illustrate problems with digital signatures: i.e. the use of digital signatures across entire documents results in signatures that can be unnecessarily invalidated by subsequent modification of the document. We propose the use of fragment signatures, which reduce unnecessary invalidation of signatures and enable greater concurrency in workflows. Our approach is document-centric and does not use a centralized database. We report on an implementation that allows fragment signatures over document fragments as well as the attachment (or embedding) of other documents. This allows collaborative or cooperative editing to occur on parts of a document without disturbing unrelated signatures. We describe the lessons learned from our deployments and offer further ways to embed such signatures into other document types. Copyright © 2010 John Wiley & Sons, Ltd.

Received 18 December 2009; Revised 9 March 2010; Accepted 16 March 2010

KEY WORDS: XML; workflow; digital signature; authorization

1. INTRODUCTION

Increasingly, document management and processing are becoming digital, in domains such as government, health care and education. Workflows for managing documents are changing from physical activities to digital processes. With this comes the introduction of new risks, particularly with respect to adherence to rules, policies and regulations. For example, physical documents in a workflow may need to be ‘signed off’ to demonstrate compliance with business rules or legal frameworks. Similar mechanisms are needed for digital workflows, e.g. for attaching signatures to documents and parts of documents for authorization.

A procurement process is an example of a workflow. In such a workflow, a request can be raised by one principal, but that request may need to be authorized by a number of other principals before going ahead. We describe the issues involved in this (and other examples) in Section 2.

Digital workflows are becoming increasingly complex. This increasing complexity leads to more flexibility in the structure of the processes in which documents are prepared, distributed and authorized. Additionally, the documents themselves are becoming increasingly free-form, i.e. that we may disregard exactly what a document requires and to write exactly what is meant by the
principal at that time. This is a common requirement: in Section 4, we introduce an analogy with paper documents; if a paper document does not exactly match the circumstances, we might choose to write something meaningful on it rather than contorting the content to match. This creates a tension with very structured digital workflows: an automated system might still be required to process the document.

As a result of this freedom, several problems related to the authorization and approval of documents and their contents arise. From content management systems to wiki articles, the contributions and changes to components that may be included in a digital document can be many and varied in their form and time of preparation. However, in the majority, current authorization systems for signatures are based on receiving authorization from participants on the whole document. If authorization is required, say for an author to give copyright permission for the use of a component of a document, the entire document must be prepared and edited before authorization of the document is completed. Further, under these systems, when changes (even minor ones) are made to an authorized document, the entire document needs to be re-authorized with all authors, regardless of whether or not the changes affect their contributions.

1.1. Structure and contribution

In this paper, we examine the use of signatures over fragment or entire documents. These documents are typically part of a larger workflow where some form of authorization or approval is required. The paper begins with three example scenarios. From these we derive a set of common characteristics and requirements for documents in these workflows. We describe an implementation of the system and report on our experiences. The paper concludes with discussions of related and future work. The main contribution is an illustration of how the use of fragment signatures within a workflow can lead to more robust signatures. This enables more concurrent work, which is difficult under paper workflows where a physical form must be passed from principal to principal, or electronic workflows using whole-file signatures.

1.2. Definitions

We start with a small number of definitions:

Document: A distinct object or artefact that can be passed or copied between individuals. It is self-contained and has a meaning (i.e. some content) as well as a presentation of that meaning. A document may be either a physical (paper) form or an electronic file.

Document fragment: A collection of one or more (not necessarily contiguous) parts of a document. This could be the entire document.

Workflow: A process of distinct steps. In this paper, workflows involve one or more documents being processed by one or more principals. The documents are copied or sent between the principals as required to meet the objectives of the workflow.

Principal: An actor or user: someone who views or modifies documents as part of a workflow.

(Digital) signature: An artefact that can be used to show that the document it is associated with has not been modified since the signature was created. The meaning of the signature has to be inferred from the document it signs, e.g. a signature on an authorization form would typically indicate that the authorization is given.

Fragment (digital) signature: A digital signature over a document fragment. For example, an authorization from a finance officer in a procurement process might not need to cover the precise specification of the equipment being purchased, but must still authorize the cost against the budget code. Thus, the fragment signature covers only the cost and budget code.

2. MOTIVATION

This section presents several scenarios to illustrate typical workflows and their documents where authorization from principals and document integrity is important, and where digital signatures
may play a role. We use these scenarios to help derive a set of common characteristics for digital workflows and a set of requirements for a system that supports digital workflows and fragment signatures.

2.1. Example scenario: Procurement workflows

A procurement process usually begins with an individual identifying the need for something to be procured. The individual might then need to complete a request form for this item. The request then passes through a number of principals:

- Line or project managers often have to confirm the requirement.
- An administrator may add further details (such as preferred suppliers and related policies, budget codes, checking that there are sufficient funds in that budget).
- A finance manager will often have to approve the request.
- Some organizations have various value thresholds: the greater the value, the higher up the management the request is passed before final approval.
- Once sufficient authority has been obtained, the order is processed. A large organization may have a dedicated unit handling procurement who then manage the request. Comments and updates could be added to the form as the process progressed to the point of completion or cancellation.

In a business-to-business (B2B) model, all the steps above can be completed electronically, and the request might go directly to a suitable supplier. In highly automated systems, production and delivery of the order might be triggered without further human check.

There are further complications. When things go wrong or as part of quality assurance processes, an audit trail can be checked. Such audit trails require tamper-proofing or they may not be acceptable to auditors. Typical features of such audit trails include digital signatures and trusted timestamps. Schneier and Kelsey [1], and later Maniatis and Baker [2], describe more sophisticated tamper-evident approaches to audit trails.

Some workflow steps could be handled concurrently, e.g. several different project managers might be able to consider a request at the same time. Such managers might also make changes or annotations at the same time and later expect to merge those changes.

2.2. Example scenario: Modules and programmes in a university

A typical university in the UK offers a number of masters programmes. In common with many universities, these programmes are organized on a modular basis, i.e. each programme comprises a number of modules.

Each module and each programme may be described by a formal document for each version, which is approved by the university’s administrative processes. These documents may be closely related because some programmes are very similar, and typically many programmes share modules. The range of overlapping modules can result in a heavy administrative burden. For example, staff cannot easily determine which programmes take their modules; programme managers cannot tell which other programmes will be impacted by a given change to a module or programme.

A common method of administering both modules and programmes involves modifying MS Word templates. Administrative changes then result in much cutting-and-pasting between old and new templates, even if there is little semantic difference. In some cases, there is no effective cross-referencing of information between the different documents until the information is entered (separately) into the administration management information system. Such systems have high potential for the introduction of inconsistencies and provide limited feedback to the principals in the workflow.

An important characteristic of such workflows is the opportunity for concurrency. Staff can work on particular documents (for modules and/or programmes) in a distributed setting. Separate approval processes in the workflow might run concurrently.
2.3. Example scenario: Event logs and case files

Database-driven command–control systems associate each ‘event’ (e.g. a request for service) with a log. This is a common model for case files: some event or issue arises, and a log is started. Comments are added, the status is updated and supporting documentation is attached. This continues until the event is completed and filed. An example of a case file is the bundle of paperwork associated with a court case.

In this example, we are considering a system model that is not centrally dependent. This allows remote, disconnected working: i.e. take the document away, work on it, return it. Typically, a central file store or database stores the definitive version of documents for users. This decentralized system differs from most implemented command-and-control systems.

A real, implemented system based on this scenario deals with some aspects of research student administration. An issue arises and a new XML document is created. Entries are added as it is dealt with by individuals. The remarks below relate to that actual system.

Arbitrary attachments are embedded in the XML documents, compressed using bzip2 and encoded in base64 form. A cryptographic checksum (SHA-1) and the attachment’s original name and length are stored to ensure consistency. Additionally, attachments can be associated with an OpenPGP detached digital signature, which is checked.

Finally, when the event is handed off (outside the system) it is packaged together automatically with a cover, narrative and all the relevant attachments and signatures to ensure consistency. Automatic processes can also manipulate the XML to add cross-references into other systems to aid in navigation of different information systems.

3. COMMON CHARACTERISTICS

The scenarios above describe the use of documents in workflows. They illustrate common concerns about both integrity and authorization in digital workflows. We separate these concerns into characteristics of documents and their operations, and issues relating to digital signatures as applied to such documents. We use these characteristics and issues to derive our requirements in the next section.

3.1. Operations on documents

The example scenarios highlight a number of common characteristics for documents in workflows:

- they can be created (usually from a template);
- are modified;
- can be e-mailed and stored;
- may contain an audit trail or history log (sometimes, this is append-only: once an entry is ‘committed’, it should not be possible to change or remove it);
- have additional documents attached (sometimes in place of part of the main document);
- can be automatically processed (for indexing, summarization or as part of a larger automatic process such as an order fulfilment system);
- be aggregated with other documents (both similar and dissimilar); and
- be exported outside the system for other use.

The characteristics above are common to individual documents. The scenarios also illustrate difficulties when using documents, in particular:

- merging documents (this depends on the logical model; sometimes, word-processing applications such as MS Word and OpenOffice.org can handle this easily);

---

1 Aside: As a design decision for this particular system, the attachments are separate XML documents to allow for multiple events to refer to one attachment with lower storage cost. Given the relatively low storage cost, this was probably the wrong decision in hindsight.
• extracting partial information automatically, or carrying out automatic processing; and
• digitally signing fragments of documents.

3.2. Digital signatures

Many documents require one or more signatures. Mostly, these signatures deal with authorization, e.g. this person may spend some funds on a purchase. However, there is an implicit need for integrity of the documents.

Many threats to the security and integrity of assets in an organization come from within that organization. Accidents can befall assets: the early detection of corrupted files is important so that they can be more easily recovered. Malfeasors or dishonest insiders already have access and can tamper with these assets. Sometimes, the motivation might not be viewed as dishonest: if a process is perceived as obstructive to the achievement of a task, workers might attempt to circumvent the process simply to complete their task. As an example, a process that requires many handwritten signatures to be collected on a collaboratively written document before presentation to a meeting encourages the splicing of pages.

The practical use of digital signatures is challenging. A typical scenario with some research paperwork is that a student might sign their form, but their signature conceptually covers only parts of the document, such as the biographical information, plan of work and an agreement to abide by the institution’s rules. People in other roles sign different parts. Essentially, we have a group of people collaboratively editing a document. Some parts are owned by specific individuals: the signature from a finance officer simply declares that the student has paid their fees. However, the conventional use of digital signatures would cover the entire document and thus be brittle in use.

This brittleness manifests itself when an inconsequential change renders an existing digital signature invalid. In the research paperwork example, the finance officer’s signature should be valid even if the student changes their plan of work. The conventional use of digital signatures, where the signature covers the entire document, falls foul of this problem. Additionally, some word processing applications make changes to document metadata when the file is opened or printed. A user who then re-saves this document may find that the subsequent signature verification fails.

A further complication of signatures is that some signatures should also cover some attachments, depending on the part of the form the attachment relates to. Finally, there has to be a suitable infrastructure for managing keys (e.g. generation, issue, use, storage and expiration).

4. REQUIREMENTS

In this section, we list our requirements for a solution to the issues characterized above. Our overall goal is to enable automatic document management where compliance with business rules or policies can be objectively evidenced (i.e. via signatures from suitably authorized principals). We have a physical analogy in mind: a paper ‘file’, with a cover sheet (the main document) plus any attachments. Additionally, a particular file could be included in a larger file, hence, this is recursive.

Our requirements, which are derived from our concerns previously described, are as follows:

Templates: Each document starts from either a blank page, or in structured systems, from a template or empty form.

View: Documents, whether empty, part-completed, or complete can be viewed. Different views are possible, depending on the media intended (print, web) or the user (perhaps only some aspects of a larger document are important). This includes taking particular extracts from documents (perhaps for sanitization when handed on).

Modify: The main document can be modified. Following our paper form analogy, we do not restrict what can be changed, provided that it conforms to the schema. Checking (via signatures, later) deals with the issue of authorization.
Audit trails/history logs: Depending on the schema, audit trails or change recording may be required. In this scenario, old data is retained rather than deleted. Presentation of change data depends on specific needs.

Signatures over fragments of documents: We require digital signatures over fragments of documents. Moreover, these should not necessarily be contiguous fragments. This reduces the brittleness of a document, i.e., changes to parts of a document that are not covered by a digital signature should not invalidate that signature.

The final complication for signatures is that we want to be able to export these documents and enable end-users to verify the signatures. This means that the signatures have to be carried with the document in such a way that they can be checked outside a centralized system with widely available software.

Attach, view, replace and remove attachments: A common component of our scenarios involves the use of attachments. Users often wish to attach other files: perhaps they prefer a different document preparation system, or the format is not easily embedded into a text document (such as a sound file, or slideshow). Thus, the attachment of documents is necessary. Importantly, these might be covered by signatures (to prevent them from being undetectably swapped later). For consistency, where multiple attachments are added, we may wish to detect and reject duplicate additions.

Merge: It may not be possible to merge documents that have been modified in parallel depending on the schemas and changes made. We do not address this further in this work.

Aggregate: It should be possible to aggregate collections of documents and process them automatically (e.g., to detect inconsistencies or create indexes).

Accessible to a range of users: Without special skills or excessive software. This means re-using existing technologies as much as possible.

Minimal centralization: We do not wish to be bound to a database or online system, although such systems might store or process these documents. In some cases, unique case or reference numbers might need to be generated and a simple solution is to obtain these from a central ‘ticket’ server. As noted above, some centralization might be needed for authorization of particular changes (although we primarily use signatures for this), such as enforcement of audit trails.

Transport/storage agnostic: The documents can be sent and stored by any electronic communication (e.g., e-mail, HTTP, stored in databases, IM).

Automatic processing: Given a document or a collection of documents, the format should enable processing by automatic tools. For example, an indexing tool might include summaries and attachment counts in its listings.

However, automatic processing is not the central purpose. We are interested in documents that may be ‘free-form’. More automated B2B systems might reasonably be more constrained.

5. OVERVIEW OF PROPOSED SOLUTION

Our desired system should satisfy the requirements above and solve the issues discussed in Section 3.2. Figure 1 illustrates our solution. Initially, a document is created from a blank template (1). This document can be viewed and edited (2). Additionally, attachments can be added (3), removed and extracted. Parts of documents (or the entire document) can be signed (4) and the signatures attached. Finally, the document can be exported so that other users can handle it (5).

This is a document-centric model of workflows. Concurrency is implicit: once a user possesses a document, they can carry out any operation upon it at the same time as other users.

We decided that documents would be stored as XML because it is an increasingly standard representation format with a wide range of tools. This increases the potential for interoperability. Alternatives include plain text (but this cannot easily represent metadata) and other document formats such as MS Word (proprietary and difficult to manipulate). We also considered OpenDocument: this is an XML-based format, but experimentation showed that some nodes were unstable under other editor operations (e.g., some attributes would change name when the document was
5.1. Signing XML

The proposed solution depends on digital signatures over XML. We justify the use of cryptography; the particular tool, in this case OpenPGP (we have mentioned the use of OpenPGP signatures already in Section 2.3); and the implications for users.

Public key cryptography (PKC) [4] is being used for two purposes: first to detect unauthorized modification of documents (e.g. preventing pages being spliced together), and second, to record authorizations (e.g. approval of a registration). Heavily centralized systems can avoid the use of PKC by recording the username of an authorizing user in audit trails, provided the authentication and recording processes are sufficiently robust. There are no other obvious alternatives that satisfy our requirements.

Our scenarios, however, require that end-users outside the system can verify the integrity and/or authorizations carried in the document. A simple assertion that a specific user made a particular authorization will not suffice. Instead, we make use of PKC signatures.

5.2. OpenPGP, CMS, XML-DSig and XML-RSig

There are two major groups of cryptographic software in common, general use. The first is OpenPGP (exemplars include PGP itself and GnuPG). The second is Cryptographic Message Syntax (CMS), implemented by (for example) OpenSSL, and used in S/MIME e-mails. Both have good toolkits openly available and are relatively well-understood. The underlying concepts are the same: both can generate digital signatures. These digital signatures should fail to verify if the signed file is subsequently altered, whether accidentally or otherwise.

Specific to XML is the XML Signature Syntax and Processing standard [5] (or XML-DSig). A major advantage of this work is that XML-DSig explicitly discusses canonicalization (more properly, the subject of other W3C standards) and the signing of parts of documents (after transformation). Additionally, there is a toolkit we can use, Sanin’s XML Security library [6].
However, XML-DSig has been criticized by Gutmann [7], principally due to problems with canonicalizing XML. An alternative, simpler version called XML-RSig, was proposed by Ernst [8] with a prototype implementation by Brooke [9]. An important lesson from these attempts is that normalization or canonicalization has to be careful and systematic to avoid false negative verifications of signatures. See Section 6.3 for further practical comments regarding canonicalization.

In the sequel, we use OpenPGP (in the style of XML-RSig), applied over fragments of XML obtained by transforms. These transforms extract only the logically relevant XML and normalize it. Notations are a useful feature of OpenPGP signatures. These are effectively a mechanism for adding a comment on a signature (which is itself covered by the signature). This means that a signature can be qualified: e.g. ‘As Finance Officer, I approve this requisition.’

A final alternative to note is the inbuilt digital signature support in applications such as MS Word (and more detailed workflow scenarios via Sharepoint-based systems) and OpenOffice.org. However, these operate on whole documents, and as such, are not useful to use in this context. We note that further efforts to develop OpenOffice.org’s support continue, as described at their wiki [10].

5.3. End-users and key management

We must also consider the end-user. Arranging key management for unsophisticated users is challenging. Some large organizations make use of X509 certificate authorities (CAs). Web servers offering the HTTPS protocol also use these certificates. By comparison, OpenPGP works on a web of trust: this is less structured than the use of CAs, although CAs can be simulated via the web of trust within an organization.

In both cases, keys need to be managed. The public keys need to be widely available so that signatures can be verified, while the secret keys are kept secure. In most cases, the intended users have basic computer skills, e.g. word processing and e-mail. Requiring them to obtain or generate keys poses a substantial problem when considering usability [11].

Thus, the interface (for low-security applications) must transparently handle issues such as key generation, the checking of web of trust or certificate paths. The user must still perform some tasks: they must decide if they will sign a document at all, and the user needs to maintain a passphrase or some form of secret credential. A centralized approach also requires the involvement of at least one CA. Applications with strong security demands clearly require higher levels of user training as part of operations security.

Additionally, the necessary software is not always easily available to end-users in tightly controlled, corporate environments. For low-security applications, a web interface to software such as GnuPG can be a good trade-off between the security requirements and usability. Two interfaces have been produced by the corresponding author for such low-security systems:

• ‘WebGPG’ allows anyone to verify a file. Additionally, keys can be generated and then used to sign files. The users have to trust that the system is correctly implemented and, that by design, it does not store passphrases or any other secrets, other than the GnuPG secret keyring. Thus, this application means that the user no longer needs to manage their own keys; for example, they do not need to ensure that they have the secret key with them when they wish to sign a file.

• ‘FileSig’ applies the model used for mailing list subscriptions to associate low-value digital signatures with an e-mail address. A file is uploaded via a web form along with the user’s e-mail address. A token is sent to that e-mail address: the assumption here is that by accessing that token, the user has control of the e-mail address. Subsequently, a signature is created.

Source code for the two implementations of XML-RSig is available at https://www.scm.tees.ac.uk/p.j.brooke/a/.

In both cases, send requests for access to the interface and implementation code to the corresponding author.
including a notation incorporating that e-mail address. This gives a small level of integrity protection (as well as time-stamping; see below).

5.4. Time-stamping notaries

If we adopt digital signatures and depend on them to indicate the time when a signature was made, we have to consider problems resulting from computer clocks. A clock could be simply mis-set, or a computer clock could be deliberately set to deceive and thus indicate that a signature was made earlier than it really was.

The problem is already known and is addressed by services such as Stamper, in RFC3161 or as in Takura et al. Thus, a simple solution is to obtain an additional signature from a trusted third-party, where that trust involves the relying party believing that the clock is correct.

6. IMPLEMENTATION

An implementation was produced based on the requirements and design presented above. The system was implemented as a web application to avoid writing platform-dependent code.

We describe the user’s perspective of the application, internal details and implementation issues of digital signatures.

6.1. User view

The user’s view is captured in Figure 1. A document can be created from a blank template, loaded from a file (XML or HTML-with-embedded-XML) or even pasted into a text box.

From the file, the application can determine the type of document and produce a suitable ‘front page’ for the document (Figure 2). This offers a range of options from viewing, editing and exporting, through to adding attachments and signatures. Furthermore, the server is not stateful: the entire document is stored in hidden fields. This reduces complexity at the server side as there is no need to record session information. Additionally, we no longer have to deal with account management on the server.

There are several important features here:

• A top banner is prepended to documents so that even when sent elsewhere, it is clear that the document can be edited and processed by XMdoc. Buttons are given linking to pages already seen in this paper.

• The ‘highlight’ buttons are used to visually indicate to the user the coverage of particular types of signatures, as discussed below.

The signing part of the page illustrated in Figure 2 offers zero, one or more fragments depending on the document and its type. In this case, ‘section1’ or ‘doc’ can be signed. If additional sections were created during editing, these would also be available for signature. When a document is viewed, the type-specific style files can be used to visually highlight the signature coverage. We discuss signatures further in Section 6.3.

6.2. Internal details

The implementation comprises both a command-line utility and a CGI application available via an Apache web server on a Linux machine. The application is mostly coded in C++ and makes extensive use of libxslt.

Individual document types are defined by a set of files. For example, the Basic_Form type comprises:

• blank.xml is a valid, blank form (essentially, the template).

†http://www.itconsult.co.uk/stamper.htm.

‡‡http://xmlsoft.org/xslt/.
about.xml and about2.xsl, which describe the type of form and identify possible signature fragments and attachment points.

• Given a document ident.xsl returns ‘MATCH’ or ‘Not’ as appropriate.

• schemawxs.xsd (and, if present, schematron.sch) define valid documents.

• name.xsl suggests a suitable name for a given document.

• attachadd.xsl, attachlist.xsl, attachremarks.xsl, attachremove.xsl and attachreplace.xsl deal with attachments. The application packages attachments in a way that can be safely handled by these XSLT files. As in an earlier example, we compress the attachments and encode them in base64 form.

• Similarly, sigadda.xsl, sigadd.xsl, sigdel.xsl, sigfrag.xsl and siglist.xsl deal with signatures. We discuss these further in Section 6.3.

• html.xsl turns the document into HTML.

• Finally, edit1.xsl turns a document into a HTML editing page. The resulting CGI form data is handled by edit2.xsl taking the original document and the form entries. This produces a new document.

It is trivial to take into account the REMOTE_USER’s identity in these scripts (although we have not demonstrated this in our current examples). This would allow different forms to be offered for different classes of user if desired.

A number of these files are generic in most examples, hence, these parts of the code are easily shared, thus reducing the burden of work when creating new document templates.

Finally, we remark that to an end-user, the web application reduces the need to install additional software. There is nothing unusual in the HTML or its limited use of JavaScript. However, we still need to arrange access to suitable cryptographic applications and consider the issues of key management, e.g. as discussed in Section 5.3. For this particular work, we used our WebGPG and FileSig web interfaces to provide the cryptographic services. We justified this in our experiments by noting that: (1) the applications have low value and (2) the usability gain by not requiring the users to install and use separate cryptographic software outweighed the security risks associated with these web interfaces. In particular, it avoided the need to users to directly generate and manage cryptographic keys. This trade-off will only apply in particular circumstances. For example, systems

Figure 2. Document page of web tool showing attachment and signature forms.
6.3. Signature fragments

Previously, we defined a document fragment as a collection of one or more (not necessarily contiguous) parts of a document (or the entire document). As our implementation is XML-based, we can more precisely define a document fragment as a (not necessarily contiguous) set of nodes selected from the document. The fragment that is created in the example (from Section 6.1) is

```xml
<xdc:frag xmlns:xdc="https://www.scm.tees.ac.uk/p.j.brooke/ns/xmdoc/common"
```

Note that there are no line breaks in this particular text: we include them for ease of printing.

Fragments are produced by running an XSLT stylesheet, `sigfrag.xsl`, that selects the appropriate nodes in an appropriate order. Use of `strip-space` and `preserve-space` directives has to be considered for each document type, as whitespace is sometimes critical to the meaning of a particular element, and at other times is totally irrelevant.

The result of the fragment production stylesheet is fed through a C14N normalizer [14]. Additionally, experimentation has shown that several additional measures are necessary. Spaces and tabs immediately before a newline are escaped (e.g. tabs become `&lt;\x09`). Finally, calls to OpenPGP applications must be in text mode to ensure that line-end conversions do not cause mischief.

Experimentation has shown that this series of measures is robust with the current implementation. The documents have been passed through mail systems, file transfers and between operating systems without invalidating the signatures (other than the point below). Such transfers can introduce problems, e.g. with line or character set encodings. However, we have not re-implemented our tools using different XSLT suites (such as Saxon or 4Suite): this would be a further test of interoperability. Similarly, re-implementing using CMS or XML-DSig (see Section 5.2) instead of OpenPGP for the cryptographic aspects might highlight other issues, such as in key management.

We do not expect as many interoperability issues, because other parts of the system, particularly fragment production, would remain the same.

The experimentation showed that one e-mail client recoded UTF-8 HTML and XML files (MIME types text/html and text/xml, respectively). As signatures are made over a series of bytes, this recoding broke the signatures. Fortunately, a trivial patch was available. It does, however, illustrate that anything other than 7-bit ASCII is always vulnerable to some form of distortion.

As documents evolve, the content of fragments can change. This places a burden on document designers: some form of version control is required so that even old documents can have the correct fragment selected.

Additionally, the overall structure of a document can change, which might rearrange the contents. The `sigfrag.xsl` stylesheet is responsible for collecting the nodes and can impose its own ordering. For example, the nodes might appear in order a, b, c in the document, while the result of the fragment could be the nodes c, b. Provided that the resulting fragment matches the fragment that was signed, any existing signatures are still valid. This feature is useful when document schemas evolve.

7. EVALUATION AGAINST REQUIREMENTS

Compared to Section 4, the solution fulfils our requirements.

Templates: Each document starts from a blank template of that type.

View: Documents can be viewed easily. CSS styling enables different views for print vs screen.

Modify: The web interface provides a separation from low-level implementation details (essentially, the XML format). Modification is allowed as per the document’s design.
Audit trails/history logs: Audit trails can be implemented, again via the document’s design. One document type implemented a simple append log. This work although suffers the obvious defect of all logs of this type: the log can be unwound until it is sent elsewhere or committed to another system (although insertion and deletion other than at the end of the log can be easily detected).

Schneier and Kelsey [1], and later Maniatis and Baker [2], describe techniques with stronger guarantees, but we have not incorporated such approaches in this work.

Signatures over parts or fragments of documents: The interface allows a selection of document fragments and formats them for robustness. The user has to take some form of action to actually sign them: this is appropriate given that the purpose of signing a fragment is to show some form of agreement, approval or authorization.

Signatures can be extracted and exported so that users who do not trust the interface can verify the fragment against its purported signatures. All the signatures and fragments for a document are included in a ZIP download.

This does not exclude the concept of signing an entire document, as in current notary services.

Attach, view, replace and remove attachments: Attachments can be added, replaced and removed via the web interface. When viewing a document, the attachment can be extracted (although this currently requires access to an XMdoc server to perform the actual extraction). Additionally, conventional detached digital signatures can be included alongside an attachment in our current tool chain.

Merge: We have not dealt with this issue and propose to deal with it in the future work. It is, however, a classic problem within metamodelling.

Aggregate: Any document can be attached to an XMdoc: thus aggregation is trivial. Less trivially, any subdocuments could be concatenated (and wrapped in an overall element) to produce a new document.

Accessible to a range of users: We have not completely tested this: other researchers and research students are the main population so far. A more formal, controlled study is being planned.

Minimal centralization: There is no central store of documents in this design. However, the use of a web application to actually process the documents imposes the requirement of accessing the web application. However, documents can be viewed off-line.

Transport/storage agnostic: The XML documents can be sent via arbitrary transports (e.g. e-mail, instant messaging). Similarly, they can be stored in databases as opaque blobs, or stored in an XML-aware database.

Automatic processing: XML has a wide range of tools available, and these can be used to program automated operations. See Section 7.2.

7.1. User experience

A prototype system was designed and implemented for demonstration in order to validate the approach and in particular whether the above requirements could be met using the standard XML technology. As such, the prototype was implemented as an example of the modules-and-programmes and event-logs scenarios described.

This prototype was implemented for the staff and students in the School of Computing at the University of Teesside. The evaluations of users with this prototype were only for purposes of demonstrating that the implementation meets the above functional requirements. Usability and user experience requirements were not evaluated formally as part of this evaluation due to the specificity of the implementation and the particularly high experience that the target user group has with technology.

As such, a full usability study is planned in the near future of a more general system for authenticating documents. That usability study will focus on the scenario regarding procurement of goods and services; it is expected to have users undertake tasks relating to the signing of contracts between multiple organizations. The users will be drawn from a pool of professionals, both novices and experts, with a variety of contract domains being represented.
From this study, it is expected that information regarding the usability of the particular implementation will be acquired, as well as possible mismatches between users’ mental models about signature forms and the realization of these forms in the digital domain.

7.2. Automated processing

Our examples illustrate the need to support automated processing of documents. A procurement workflow might be highly constrained, following a tightly defined process. Similarly, university modules and programmes have a formalized structure in terms of title, credits, prerequisites, yet the details of the content might be quite free-form. The least structured of our examples concerns event logs and case files, yet even these have some notion of status and indexing.

Thus, we have to acknowledge the tension between very structured documents and the user’s desire to write exactly what they mean. Our analogy from Section 1 is that if a paper document does not exactly match the circumstances, we might choose to write something meaningful on it rather than contorting our words to match the document. Hence, boxes might not be properly ticked and additional explanatory text might be written into other parts where an automated system might disallow or ignore this text.

At the other extreme, highly structured systems can enable automated order fulfilment with little human involvement. Failure to verify a signature might cause the process to stop and require human intervention. A middle-ground concerns documents with some highly structured parts, and other less structured, more narrative parts. These can be indexed, but not necessarily processed in detail.

The main constraint then becomes the specific document schema. Attachment data is necessarily opaque, as is the actual signature data. But the metadata is not opaque and can be easily processed. The current tool chain has features for listing and extracting both signature and attachment data.

A useful side effect is that we could check that a specific individual has signed specified parts of the document and verify this signature.

Further investigation of the balance between structured documents and human flexibility is needed. A common part-solution is to add ‘escape hatch’ fields that may contain arbitrary additional information. But note that these cannot be processed automatically.

7.3. Scenarios revisited

We revisit our motivating scenarios from Section 2.

Procurement scenario: A digital procurement workflow may already use structured digital documents. Our approach would enhance this by allowing fragment signatures which are fine-grained. The trail of updates from the procurement unit would, for example, be only related to the document by a unique reference number. Appending further updates would not damage any earlier authorizations. However, an update to the request that changes, say, the number of widgets to be supplied, would necessarily cause signature verification to fail. Generally, the authorizations could be trivially checked by checking the signatures against the relevant document fragments.

Modules-and-programmes scenario: The modules-and-programmes example is much more complicated than the procurement example, in that there are many interacting documents. Individual structured XML documents describe either a module or a pathway. Automation is clearly necessary for the consistency checking intrinsic to this particular problem.

Our approach enhances such workflows by enabling fragment signatures that apply to most or all of a document, such as final approval from a committee. At the same time, other fragment signatures might only cover subsections, such as confirming that the necessary IT resources are available. For some parts, this means that easy translation to newer formats is available and merely cosmetic template changes are irrelevant to the signatures.

Event logs and case files: The final example dealing with event logs and case files is the most general. As in the previous two scenarios, our approach enhances these workflows by using fragment signatures to allow greater authorization and approval control over parts of the document while reducing brittleness and allowing parallel work.
In all three cases, a web front-end is helpful for users: we cannot expect users to edit raw XML. In other cases, updates from other tools (e.g. from e-mails) could be used to automatically apply updates.

8. RELATED LITERATURE

We encounter two other approaches to document management

- Database-driven systems with an audit trail. These are common in centralized models and the audit trail is implicitly trusted along with the database. Electronic exports from such a system can carry a digital signature across the export asserting that the export is a true record. These are then essentially read-only.
  
  Access to such systems often depends on continuous network availability: it is difficult to take away parts of a database, work on them remotely and then merge the changes. Moreover, they are typically more regimented and structured. Thus, such systems are not especially flexible or free-form.

- Use of documents with a digital signature across the entire document: but we have rejected this at the outset due to the brittleness of these signatures. Additionally, a signature across an entire document means that any change will invalidate that signature—this is a core part of our motivation.

There are other options for dealing with attachments. At its simplest, we could simply ZIP files. OLE objects could be leveraged: Microsoft’s Binder product effectively packaged multiple objects. This concept appears in other applications, such as word processors, where distinct objects can be embedded either by reference or by value. An important distinction of our approach is that it allows signatures to encompass attachments.

Security and integrity in collaborative systems arises broadly. A typical example is that posed by Wikipedia. Here Jensen [15] describes the issue of integrity and discusses a method to prevent untrusted authors from modifying other documents. Although Jensen uses a reputation system, it is interesting in that it illustrates the problem of integrity.

Similarly, Krowne and Bazaz [16] also look at CSCW and describe two authority models, free-form and owner-centric. The model adopted in our work is closer to the free-form model. Also related is the work of Miklau and Suciu [17], who discuss integrity issues in web publishing. Again, signatures arise here, and issues of what a signature means have to be addressed. We suggest that the use of OpenPGP’s notations is helpful here, as described in Section 5.2.

Bertino et al. [18] is another variant dealing with distributed and collaborative updates. However, this is a relatively complicated approach that specifies the sequence of subjects who should handle a document. We take a laissez-faire approach to who actually handles a document, instead preferring to deal with it via the authorization model implicit in the signatures collected on a document.

In Section 5.2, we discussed our choice of cryptography toolkit and described related issues, notably that of normalization or canonicalization. Using XML is an easy choice for many new systems: there is a wide range of tools that support processing and transformation (in our case, we used libxslt). Lautenbach [19] gives an overview of this area. However, Gutmann [7] criticizes XML-DSig [5]: we discussed this further in Section 5.2.

Avoco Secure [20] reports important aspects in signature software. They deal with some similar matters (e.g. concern about time-stamping). Particularly interesting is the first remark on handling dynamic content. The example here is the use of dynamic fields such as updating date/time. In our approach, we simply exclude such fields from a document fragment, or if it is semantically important, we fix it and include it in the signature (in which case it should not be updated any further).

Berbecaru et al. [21] start by remarking on the European Directive on electronic signatures, then survey the field of electronic signatures and propose a layered architecture. Unsurprisingly, they deal
with similar issues, including independent and nested signatures. Berbecaru et al. classify signatures as enveloped, enveloping and detached, as do Lioy and Ramunno [22]. In this classification, most of our signatures are enveloped, that is, they are embedded in a document. Lioy and Ramunno deal with multiple signatures over ‘dossiers’ of documents and conclude that there are aspects to be specified further. We suggest that our work is a practical implementation.

Kain et al. [23] sound a cautionary note indicating that signatures are just part of the security problem. In this work, examples are given where a signed document is presented in such a way so as to deceive recipients. This is clearly a valid concern. In our approach, we aim to mitigate this problem by allowing re-verification and, importantly from a user perspective, highlighting parts of documents that are covered by a signature. However, problems could still be ‘smuggled in’ via embedded, opaque attachments.

There are some interesting potential extensions which we do not address. For example, Bull et al. [24] use ‘content extraction signatures’ to ‘enable selective disclosure of verifiable content’. This is a combination of privacy as well as conventional integrity/authenticity issues.

9. LESSONS LEARNED

We describe the lessons learned from various deployments based on the events logs and case files scenario (Section 2.3). The deployments managed documents relating to the complicated training processes for individuals in a large organization. There were several hundred case files covering around 50 individuals with multiple attachments.

These deployments highlighted the points below, as well as those described in Section 6.3, i.e. that some files can be corrupted by transmission resulting in signatures being broken, and that documents evolve.

The implementation broadly matches the requirements described in Section 4 although our choice of a web-server mediated tool means that some centralization is inevitable. More refined (but platform-dependent) tools could reduce this problem.

This experience shows that the design of document models, templates and schemas is important. Some change to underlying schemas is inevitable as organizational processes evolve: when changes have been required, we have found that relatively simple XSLT transforms have sufficed to automatically update existing documents. This can preserve signatures, provided that the relevant nodes are unchanged (as discussed in Section 6.3). The separation of visual representation from the logical structure is one aspect of this.

The main difficulties surround the brittleness of signatures. This is a significant part of our motivation. By necessity, we desired relatively free-form documents both in terms of workflow and layout. This relatively unconstrained problem allows lots of scope for introducing inadvertent changes that, while semantically insignificant, cause false bad signatures. Additionally, we opted for XML-based documents. In Section 6.3, we identified the measures taken to make these signatures more robust.

The more the content of a document that is covered by a signature, the more likely it is that the signature will be invalidated by a change to that content. Hence, signatures over fine-grained parts of documents are important.

Consideration of the meaning of a document is critical to this: in some cases, it is questionable as to what purpose a signature actually serves. Hence, reducing a signature to cover only as much as is necessary helps to improve the overall robustness of these signatures. The creation of fragments by selecting appropriate nodes in an appropriate order is a mechanism to aid this reduction of coverage.

By selecting nodes, we found that by consideration of the meaning of the node, further transforms could be applied. A simple example is the removal of unnecessary whitespace in a ‘title’ field (e.g. ‘(space)(tab)Ms’ is transformed to ‘Ms’). Another example is that case-insensitive fields can be normalized to, say, upper case for the purpose of signing. More complex conversion depends on the specific meaning of a field within a document.
Separately, some form of ‘escape hatch’ is often needed (see Section 7.2). Forms do not fit all circumstances properly and need some way to indicate additional information. The addition of fields for arbitrary text and attachments is such an escape. Depending on the expected usage, this may or may not be covered by particular signatures.

10. CONCLUSIONS

We have presented an approach for document-centric workflows with fragment digital signatures for purposes such as authorization. The types of document are motivated by real-world scenarios and the model is applicable to a wide range of such scenarios including computer-supported cooperative work in general. The model has been validated by a small-scale deployment in real use.

This work principally deals with integrity and authorization issues by embedding digital signatures over fragments of the documents. The embedded signatures enable simpler workflows by reducing the brittleness of digital signatures compared with normal usage (i.e. signing entire files). They are more robust due to the use of transformations and canonicalization. In particular, they allow for fine-grained signing of parts of documents. These signatures are important as they often indicate authorization or some form of approval. In some cases, this enables more concurrency within a workflow by allowing principals to modify and sign only the parts relevant to them.

These fragment signatures are important: they enable better collaborative working on documents without compromising signatures unnecessarily. This more flexible use of signatures still allows the detection of inappropriate alteration: a signature covering a particular document fragment will fail to verify whether that fragment is modified. Aggregation of multiple signed subdocuments into a new document becomes possible. Export from a workflow is supported: the underlying XML document, a ‘clean’ presentation, attachments, document fragments and signatures are all included. This allows external verification of signatures and arbitrary processing elsewhere.

For usability, attachments can be carried within the document as users often wish to embed or attach other types of document to a typical form. To demonstrate the features described in this work, we have developed a web-based tool implementing the necessary features.

10.1. Future work

An obvious extension of our work is to enable encryption of nodes (and their subnodes) to deal with secrecy. This is much simpler than signing because the normalization issues do not arise; but it becomes harder at the user interface in determining when an encrypted node should be saved in a decrypted form. Prior to this work, we will need to identify appropriate use cases to guide the feature design.

Re-implementation of our tool to use different XSLT suites might highlight interoperability issues. Similarly, replacing the cryptographic aspects with CMS or XML-DSig could illustrate other problems. A further alternative is to write custom XSLT extensions to perform the specialized operations, e.g. verifying the signatures in a document.

We initially considered implementing our solution as an embedding within the OpenDocument format [25]. A prominent application using this format is OpenOffice.org. By this embedding, we would hope to leverage the powerful editing facilities already in OpenOffice.org for the user. At the same time, web-based, GUI or command-line tools could modify the signatures and any attachments.

However, experimentation discovered that additional files are not retained in ODT (OpenDocument) files. Therefore, these additional files would need to be embedded deeply within the XML structure of the ODT file. Additionally, some nodes (such as attributes describing styles) are unstable. These problems caused us to put aside OpenDocument. However, the interface of OpenOffice.org and similar applications is compelling for users [10], hence future work would attempt to overcome these difficulties. Other interfaces to consider include web browsers: for example, Mazumdar [26] describes an extension for Firefox that allows XML signature processing.
Finally, and most generally, it would be useful to address the balance between highly constrained computer documents and humans using paper processes in a flexible fashion (see Section 7.2). We propose addressing this through future usability testing.

ACKNOWLEDGEMENTS

We thank the anonymous referees for their constructive comments.

REFERENCES

