

Bulletproof Business Process Automation: Securing XML Forms with Document Subset Signatures

John M. Boyer, Ph.D.
PureEdge Solutions Inc.
jboyer@PureEdge.com; jboyer@acm.org

ABSTRACT

The standard unit of work in the business process is the electronic form, which includes complex user interface designs, data gathering and validation, wizard-like behaviors and spreadsheet computations. This paper reports state-of-the-art digital signature methods used to help provide security, non-repudiation and auditability within complex electronic forms applications. Both for financial reasons and for compliance with government regulations such as the Government Paper Elimination Act (GPEA), an ever-increasing number of intricate electronic forms applications are being created. Yet, there are aspects of paper-based systems that can only be modelled if digital signatures are able to omit carefully specified portions of a document so that certain *restricted* changes can be made after a signature is affixed. These scenarios occur frequently with electronic forms that are multiply signed and possibly involved in a non-trivial workflow process.

Due to the importance of securing electronic forms and the complexities that can arise in signing them, the W3C XForms working group has placed integration with XML signatures among the *highest priorities* of its new standardization charter. However, there are security issues that must be addressed but which are beyond the core cryptographic capabilities of the W3C XML Signature Recommendation. This paper presents our research into and solutions for these issues. The purpose of this paper is to let successful industry experience and academic analysis provide guidance to future secure document standardization efforts (such as XForms) so that, even in the most demanding signature scenarios, the standards are still able to meet the fundamental requirement of digital signatures: *What you see is what you sign*.

Categories and Subject Descriptors

K.4.4 [Computers and Society]: Electronic Commerce—*security*; I.7.2 [Document and Text Processing]: Document Preparation—*Markup languages, Standards*; G.2.3 [Discrete Mathematics]: Applications

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM Workshop on XML Security. October 31, 2003, Fairfax VA, USA.
Copyright 2003 ACM 1-58113-777-X/03/0010 ...\$5.00.

General Terms

Security, Legal Aspects, Algorithms, Standardization

Keywords

Secure XML Forms, XML Signature Transforms, XForms, XFDL

1. INTRODUCTION

The corollary of business process automation is growth in the volume and value of electronic commerce transactions, along with a corresponding increase in security expectations. The prevention of on-line fraud is facilitated by security methods that integrate easily and seamlessly across applications and organizations. Due to its emphasis on interoperable open systems, XML [7] has been heralded as a key enabling technology for business process automation and e-commerce. The World Wide Web Consortium (W3C), which publishes the XML Recommendation, has also published companion recommendations such as XSLT [10] and XML Signatures [14] that assist in the development of business process automation and e-commerce systems which are interoperable and secure.

In many of its emerging roles, XML needs to be secured [13]. The XML Signatures Recommendation [14] describes a syntax and processing semantics for creating and validating a digital signature over both XML and other data. XML signatures are capable of satisfying all requirements of the Digital Signature Guidelines of the American Bar Association [1], which states that security is achieved via message and signer authentication and signer authorization. Message and signer authentication are achieved via hashing algorithms, public-key cryptography and PKI. Signer authorization refers to the fact that the signer authorizes the transaction covered by the digital signature, which implies that the message covered by the signature must not omit information pertinent to the interpretation of the transaction record. In other words, within the application context in which a signature is presented, the message signed by the signature must capture not only the answers to questions, but also the questions themselves along with all relevant user interface properties such as font size and color, background color, visibility status, and the positions and types of user interface elements [3]. This concept is embodied in Assertion 1.

ASSERTION 1. *Digital signature security relies on the ability to accurately reproduce the user interface presented to the signer from the cryptographically secured message because this is what the signer believes is being signed. In essence, what you see is what you sign [14].*

Since separation of presentation and data is desirable, a single XML signature can cover multiple resources so that the full context of the signed information can be protected by the signature. Alternately, XML permits data and presentation to be kept in separate parts of the same file (e.g. see XForms [12]).

1.1 XML Document Filtering: Definition and Practical Scenarios

DEFINITION 1. A *document filter* computes a subset of a given document.

DEFINITION 2. A *document subset signature* is a digital signature associated with document D that applies a document filter to D to obtain the message to be cryptographically protected.

The XML Signature Recommendation provides document filtering¹ via XPath expressions [6, 9]. The serialization of the document subset computed by a document filter is used as the ‘message’ to be signed. Before delving into the security implications of document filtering, it is helpful to consider the motivating scenarios that come up in practice.

Forms can be described as “the primary user interface—the visual record—when transacting electronic commerce” [11]. Industry analysts estimate that organizations worldwide produce 90 billion pre-printed pages per day—a significant portion of which are paper forms [2]. Clearly, paper forms are a vital part of the business process. However, paper-based systems are inefficient and costly in a digital world. Analysts have estimated that it costs organizations in the USA over \$120 billion annually to process paper forms [8]. Yet, paper forms have certain inherent properties that are challenging to model in electronic forms without document filtering.

Many electronic forms scenarios that require document filtering also require multiple digital signatures. As an interesting example of deployed software, consider a form that must represent a joint legal filing. Just as with a paper filing, the electronic form of the joint legal filing must remain flexible about the number of parties to be added and permit the addition of their demographic data and supporting document attachments, yet the signature of each party must not permit modifications to the essence of that party’s contribution to the form.

Similar examples abound in insurance applications, financial documents, and filings of compliance with government regulations. In general, multiple signature scenarios in which both data and user interface elements must be added after a signing event are of greatest interest in this paper.

1.2 XML Document Filtering: Security Issues

Due to the importance of securing electronic forms and the frequency with which complex signing scenarios arise

¹Readers new to this area may wonder why international standards bodies like the IETF and W3C would define document subset signatures rather than, say, signing the document at a given time then recording a ‘diff’ for each subsequent change. The answer is **security**. The document filter specifies the ‘diffs’ that can be made after the signature is applied, and any non-permissible ‘diffs’ invalidate the signature. Moreover, the document filter is signed, so the set of permissible ‘diffs’ cannot be altered.

for electronic forms, the XForms working group, the current W3C standardization team for electronic forms, has placed integration with XML Signatures to be among the highest priorities in its new standardization charter. As the XForms working group begins to consider how to incorporate digital signatures into electronic forms, it will be necessary to integrate with XML Signatures in a way that allows the solutions presented in this paper to be applied. Fortunately, one of the most promising aspects of XForms is its reliance on a ‘host language’ to provide the user interface and certain related services. This allows XForms to more easily interoperate with existing XML languages such as XFDL [5] that are designed and implemented to handle the intricate signature scenarios reported in this paper. To precisely identify the security issues, we need some additional definitions.

DEFINITION 3. The term *item* refers to a user interface element or to the XML element² that represents it in a document.

DEFINITION 4. A *signed item* is an item that is included in the hashed message of a digital signature.

DEFINITION 5. An *unsigned item* is an item that is excluded from the hashed message of a digital signature.

In signatures over XML forms, document filters are used to distinguish signed items from unsigned items. When a form must be filtered before signing, it is typically a best practice to use omission logic rather than inclusion logic to express the document filter [4]. An *omissive expression* explicitly identifies the characteristics of the portion of the document that will not be signed (e.g. using the `not()` function in XPath). However, if unsigned items can impact the correct interpretation of the signed information within the application context, then Lemma 1.1 shows that the resultant signatures are a disservice to the users of the application despite the incorporation of the best cryptographic methods [4].

LEMMA 1.1. An item I can be modified in, deleted from, or added to document D without invalidating an XML signature S over D if and only if the document filter for S classifies I as unsigned.

PROOF. An XML signature signs its document filter, so the filter applied during signing cannot be changed afterward. During signing, the document filter takes a subset of D , the serialization of which is the message M over which the signed hash is calculated. During validation, the document filter computes the subset M' of D whose hash is compared for equality with the hash of M protected within S . If the document filter classifies I as an unsigned item, then modification, deletion or addition of I to D does not change M' relative to M , so S is not invalidated by the change to D . On the other hand, if the document filter classifies I as a signed item, then addition of I to D adds I to M' , which invalidates S since I is not in M . Likewise, modification or deletion of a signed item I from D changes M' relative to M , invalidating S . \square

²In XML [7], an *element* consists of a named bracketing tag set, attributes, and internal content.

Given Lemma 1.1, adherence to the signer authorization principle³ requires that the document filter expression should be as exacting as possible in characterizing the items to be omitted. However, in XML Signatures [14], we are constrained to the information available in the XPath data model [9]. Yet due to arbitrary application-specific semantics, information dependencies among items can arise not only from node content but also from intermediate information not available to the data model, such as that obtained from graphical rendering systems. This paper focuses on two algorithms that augment the sign and validate events of document subset signatures to increase their security according to Assertion 1. The statement of problems in this paper requires the following definitions, and the solutions to these problems represent new best practices in the security of electronic forms.

DEFINITION 6. A *bounding box* is the smallest enclosing rectangle of the user interface element associated with an item.⁴

DEFINITION 7. A *grouping item* is an item that visually binds other items with its border and background effects (which render around the bound items and in their transparent regions).⁵

DEFINITION 8. A signed non-grouping item is *obscured* if any portion of its bounding box overlaps with that of an unsigned item.⁶

DEFINITION 9. A signed item has undergone a *significant layout change* after signing if

1. the horizontal or vertical position of its top-left corner has changed relative to the top-left corner of any other signed item.
2. the horizontal or vertical position of the bottom-right corner has changed relative to the top-left corner of any other signed item.
3. any portion of its bounding box shifts to or from negative pixel space.⁷

³The message covered by a signature must not omit information pertinent to the interpretation of the transaction record

⁴Bounding boxes are taken to be the suitable descriptors of *business form* layout because the vast majority of form items have rectangular user interface elements. The few items that do not (e.g. items representing radio buttons) still typically have rendering control over a rectangle, both in forms languages and in the underlying windowing subsystem.

⁵For examples, see 'div' tags in HTML [17] and 'box' items in XFDL [5]. In XML languages designed not for securing *business forms* but rather for rendering broader classes of documents (e.g. SVG [15]), identification of grouping items is more involved due to complex clipping and transparency rules.

⁶The assumption of item opacity (i.e. **non-transparency**) is appropriate since overlapping pairs of *signed* items are not considered to be obscuring one another by this definition. When two or more items are deliberately overlapped with the intent of using clipping and transparency rules to combine the items visually, then the items are logically one unit, so they should all be signed together.

⁷Negative pixel space is the area to the left of the left margin and above the top margin of the form.

Based on these definitions, we are able to state *as requirements* the problems for which this paper provides solutions. Theorem 1.2 proves the necessity of performing an overlap test to detect and forbid obscured signed items during both signature generation and validation. Theorem 1.3 proves the necessity of performing a layout test to help prevent significant layout changes among signed items.

THEOREM 1.2. An augmentation is **required** to both the sign and validate events of a document subset signature S over an electronic form to prevent signed non-grouping items from being obscured.

PROOF. By contradiction, suppose the signing event for S permits an obscured signed non-grouping item I_s , which could represent extra terms and conditions of an insurance policy or other contractual agreement. The obscuring unsigned item I_u could be a simple empty label with the background color of the form, or it could contain alternate terms and conditions that are more favorable to the signer. Once S is affixed and the electronic form is delivered to the validator, I_u can be moved or deleted, and according to Lemma 1.1, S remains valid yet I_s is unobscured, which could encumber the signer to extra or less favorable terms and conditions, violating Assertion 1. Thus, the signing event must not permit obscured signed non-grouping items.⁸

Given that an obscured signed non-grouping item I_s is not permitted during the signing event, we now suppose for the sake of contradiction that I_s is permitted by the validate event for S . The obscuring unsigned item I_u must have been modified in or added to the electronic form after signing since the signing event forbids I_u . According to Lemma 1.1, the modification or addition of I_u does not invalidate S , yet I_s is obscured to the validator, which again violates Assertion 1. Thus, the validate event must not permit obscured signed non-grouping items.⁹ □

THEOREM 1.3. An augmentation is **required** to the sign and validate events of a document subset signature S over an electronic form to prevent significant layout changes to signed items.

PROOF. Once a signature S is affixed, significant layout changes among signed items can cause:

- an overlap between signed items that did not exist at signing time
- remove an overlap between signed items that did exist at signing time

⁸Note that a technology that violates this tenet is as much a disservice to the validator (e.g. an insurance company) as it is to the signer (e.g. an insurance applicant). The unscrupulous validator scenario is sufficient to prove the theorem, but given an honest validator, an unscrupulous signer could repudiate S by simply demonstrating the ability to obscure signed items during the signing event.

⁹One might at first find the argument for the signing event more compelling because the final document does not contain any trace of I_u . However, the validate case cannot be deemed less problematic because I_u must appear in the final document since the next logical step is that the document must be searched for I_u , which is precisely what the theorem claims is necessary.

- change the relative order of two signed items with respect to a natural reading order
- change the size of a signed item, which may affect its appearance or the information it displays
- alter the visibility of signed items by changing whether they are in negative pixel space.

The list covers all conditions of Definition 9, and each point can result in an altered interpretation of a signed item, violating Assertion 1. Thus, to prove the necessity of an augmentation to the validate event, we only need to prove that a significant layout change can occur without disrupting the core cryptographic validation of S .

Languages for expressing electronic forms (e.g. [5, 17]) permit the use of relative positioning of items, which may be explicit (e.g. the ‘align’ attribute on the IMG tag in HTML [17]) or implicit (e.g. the normal item position in HTML is after the predecessor, or wrapped to the left margin and below the predecessor). If signed items are relatively positioned to unsigned items, then it is possible to change the layout among signed items by changing the unsigned items, which can be done without breaking the core cryptographic validation by Lemma 1.1. Consider a document containing items w , x , and y in a row positioned left to right. Let the language define the terms *after*, *below*, *before* and *offset* with the obvious semantics. Let the document achieve the left-to-right order of w , x and y by defining x to be after w and y to be after x . Let signature S omit x . After affixing S , change x to be ‘below’ w . This changes the relative position of w and y . Alternately, if x is placed ‘before’ w , then y would overlap w . If x were then ‘offset’ further into negative pixel space, then y would follow it and become invisible to the user. Finally, let the language also define the notion *expand right to right*, and add z ‘below’ w and ‘expand right to right’ with y . Affix S , then change the width of the unsigned item x . This move y relative to w but it also changes the size of z .

The necessity of an augmentation to the signing event for S is proven by considering how the validate event augmentation will prevent significant layout changes. The examples above demonstrate that the signed information is insufficient. Hence, the validate event augmentation requires further data to detect significant layout changes, yet this data must be added to S because only what is signed is secure [13, 14]. \square

Although XML document standardization groups (such as the XForms working group) are only just beginning to consider how to integrate with the XML Signatures Recommendation, Theorems 1.2 and 1.3 show that the security demands of complex signature scenarios will require an integration that permits the application of overlap testing and a test of significant layout change after the core validation of a digital signature. Furthermore, in order to maintain the **scalability** of document processing systems, efficient algorithms that achieve expected $O(n)$ performance will be required since this is the performance characteristic of core signature validation.

Section 2 describes an efficient algorithm that finds all pairs of overlapping items and forbids the overlap of a signed item and an unsigned item. A number of practical issues are discussed, including performance requirements as well as a few special cases in which overlap between a signed item and an unsigned item is permitted and in fact preferred. Section 3

describes an efficient test to detect changes to signed items’ relative layout in positive pixel space. Changing a signed item to or from negative pixel space is implicitly detected.

2. OVERLAP TESTING

In compliance with the requirement of Theorem 1.2, the overlap test determines every pair of items (user interface elements) that share a common geometric region (rectangle), then the test determines whether the pair consists of one item signed by a signature S and one item not signed by S . The overlap test could be performed once for each signature, but the complex electronic forms and related documents typically deployed by government and industry contain several hundred to several thousand items, and often contain multiple signatures and multiple pages. Therefore, to optimize the valid case, it is best to implement the test such that it is possible to perform the overlap test once (on each page), then test each overlapping pair against all document subset signatures. Section 2.1 presents the overlap test as an extension of the classic computational geometry algorithm for detecting intersections among a set of rectangles.

While the expected complexity of pages provides one reason why the overlap test must be made as efficient as possible, the test must also be performed in a number of scenarios in which undue delays cannot be tolerated. For example, an overlap test should be performed when a document is first brought up, when switching pages, and when a layout change occurs on the current page due, for example, to the addition or deletion of items. When the user affixes a signature, only the new signature must undergo the overlap test, but the test must not cause undue delay. Thus, an $O(n^2)$ brute-force comparison of every pair of items is unacceptable. Since the creation or validation of a digital signature is $O(n)$, the overlap test solution should at least achieve expected $O(n)$ performance per signature. Section 2.2 presents further implementation details and a proof that they result in the desired performance profile. Finally, Section 2.3 presents an interesting additional benefit of overlap testing for protecting empty regions near signed items.

2.1 Overlap Testing by a Plane Sweep for Rectangle Intersections

A classic strategy used to solve many basic computational geometry problems is the ‘plane sweep’ method. Typically, points of interest on a plane are sorted by their x -coordinates, and a data structure is used to represent a ‘vertical sweep line’ that is moved from left to right, stopping for updates at each x -coordinate point of interest. Alternately, the points of interest could be defined by their y -coordinates, and the data structure containing the objects under consideration would be a horizontal sweep line. The example in Figure 1 uses the latter method in order to illustrate the approach that should be used when the human language presented by the user interface flows horizontally (e.g English).

The rectangle of an item’s bounding box can be defined by the coordinates of its top left corner (x_1, y_1) and its lower right corner (x_2, y_2) . Clearly, two rectangles do not overlap unless both their x -intervals and their y -intervals overlap. The x -intervals of items I and I' overlap if either $(I.x_1 < I'.x_1 \text{ and } I.x_2 \geq I'.x_1)$ or $(I.x_1 \geq I'.x_1 \text{ and } I.x_1 \leq I'.x_2)$. The y -interval overlap testing is performed implicitly by the sweeping action of the algorithm:

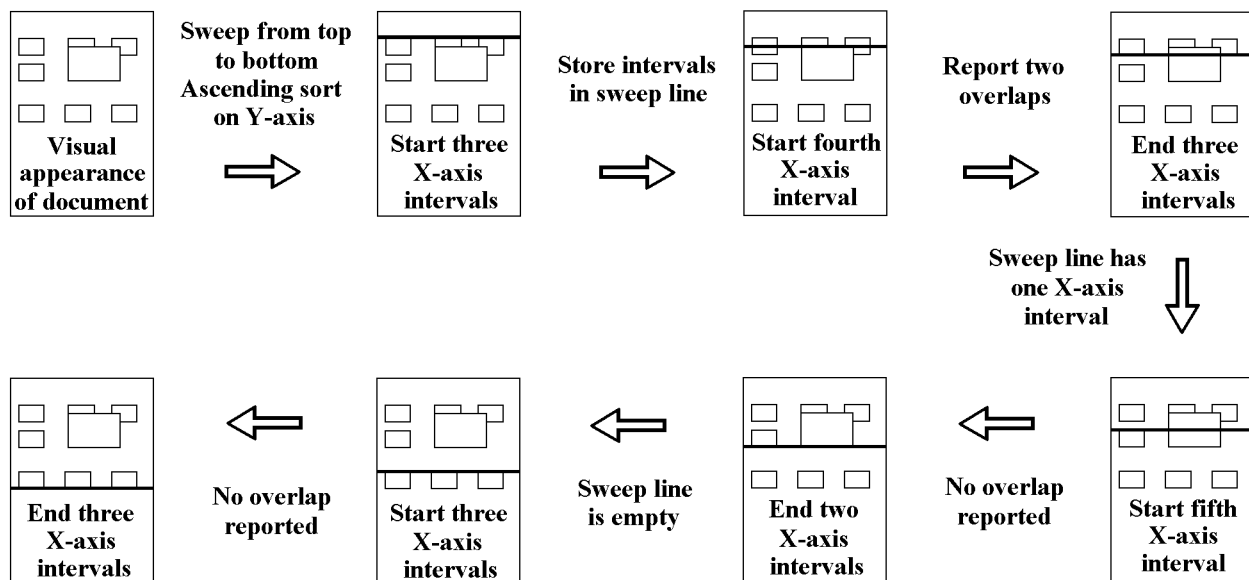


Figure 1: Example States of the Overlap Test Processing Model

1. For each item I , define a *start event* for $I.y_1$ and an *end event* for $I.y_2$.
2. Sort the start and end events by the y -coordinate, breaking ties by putting start events before end events and arbitrarily otherwise.
3. Create an initially empty horizontal sweep line L .
4. For each event e in sort order,
 - (a) If e is an end event for item I , then remove the corresponding start event for I from the sweep line L .
 - (b) If e is a start event for item I , then
 - i. If L contains an event whose corresponding item I' has an x -interval overlap with the item I that corresponds to e ,
 - ii. and if one of I and I' is signed and the other is not,
 - iii. and if the signed item is not a grouping item,
 - iv. and if the overlap exceeds a tolerance¹⁰ specified by the signed data,
 - v. then **invalidate the signature**.
 - vi. Insert e into the sweep line L .

Figure 1 presents an example of the geometry algorithm. The first state at the top left represents the visual appearance of the document. The next state shows the sweep line L starting with the first three rectangles. The third state

shows the sweep line advancing to the start event of the fourth, larger rectangle. Since L has not reached the end events for the second and third rectangle, they are found to overlap the fourth rectangle. The fourth state shows L reaching the end events for the first three rectangles, which leaves L only containing the fourth rectangle. The fifth state on the bottom right of Figure 1 depicts the start event of the fifth rectangle, which is added to L without reporting an overlap because its x -interval does not coincide with that of the fourth rectangle. The sixth state shows the end events for the fourth and fifth rectangles, which leaves L empty. The last two states show the start and end of three more rectangles. Theorem 2.1 establishes the correctness of the overlap test algorithm.

THEOREM 2.1. *The overlap test algorithm is a sufficient augmentation of the sign and validate events for a signature S to satisfy the requirement of Theorem 1.2.*

PROOF. That the overlap test algorithm finds all rectangle intersections is a well-known result proven in [16]. The tests in step 4.b.ii and iii ensure that a signed non-grouping item has been obscured. For strict conformance with Definition 8, let the tolerance in 4.b.iv be equal to zero and unchangeable by the document author. \square

Remark: The definition of obscuring a signed item deliberately permits a signed grouping item to be overlapped by unsigned items, including other grouping items. In some documents, a large grouping item G_1 contains some signed items plus a smaller grouping item G_2 , which contains unsigned items (e.g. an ‘office use only’ section of an electronic form). It is necessary to sign G_1 to ensure that the signed items within G_1 will have the proper background color. However, G_2 could be changed between visible and non-visible or be moved around inside G_1 , but it can only overlap *empty* spaces within the signed grouping item. To prevent this, the empty spaces must be secured, which is discussed briefly in Section 2.3.

¹⁰To faithfully reproduce the appearance of a tightly packed paper form, the items of an electronic form often must overlap by the amount of space allocated to their borders (typically two pixels). In some business forms, industry experience (of the author) has shown that it is necessary to permit slightly more tolerance. The amount of tolerance should be configurable by the document author who, by the very act of configuration, is making a security choice based upon the nature of the document (e.g. a six pixel tolerance can often achieve the desired visual effects while not materially affecting the information content of the obscured items).

2.2 Implementing the Overlap Test to Achieve $O(n)$ Expected Time on Electronic Forms

The running time of the general rectangle intersection problem is typically reported to be $O(n \log n + s)$, where n is the number of rectangles and s is the number of intersections [16]. However, there are properties specific to electronic forms and similar documents that permit a faster implementation as shown by Theorem 2.2.

THEOREM 2.2. *On electronic forms, the overlap test determines whether to invalidate a signature S in expected $O(n)$ time, where n is the number of items.*

PROOF. Since there are $O(n)$ events (two per item), step 1 is clearly $O(n)$. Normally, sorting the y -coordinates in step 2 would take $O(n \log n)$ time. However, the sort keys are screen coordinates, which are small integers,¹¹ so it is not costly (relative to the several hundred to a few thousand items on a page of a typical form) to use a bucket sort with a bucket for each screen coordinate. Step 3 is clearly $O(1)$, and Step 4 performs $O(n)$ iterations of its loop body, which must therefore achieve constant time.

Electronic forms are virtually always restricted in the number of items placed horizontally¹² because users virtually never accept applications that require heavy use of a horizontal scroll bar. When one also accounts for the typical width of items, the number of rectangles appearing simultaneously on the horizontal sweep line can safely be assumed to be a reasonably small constant¹³. Given the assumption of a constant-sized L , steps 4.a, 4.b.i, and 4.b.vi are $O(1)$ even if implemented by a simple array or linked list with sequential search, add and delete operations.

Steps 4.b.ii to 4.b.v are specific to the overlap test (i.e. not part of the classic rectangle intersection problem). To efficiently implement step 4.b.ii, a preprocessing step is used to create the document subset for each signature. Membership queries can then be performed in constant expected time if the document subset is backed by a hash table keyed with the memory pointer values of the document's parse tree nodes. The running time of step 4.b.iii is language-dependent, but the identification of a grouping item can be done in constant time in languages designed for security (e.g. in XFDL [5], an item is a grouping item if and only if its element tag is 'box'). Step 4.b.iv is an $O(1)$ requiring only a few logical and arithmetic operations, and Step 4.b.v requires setting a simple Boolean flag. \square

Remark: It is typically more efficient to perform the tolerance test and the grouping item test, then the test for whether one item is signed and the other isn't. The reason is that these tests eliminate all of the detected overlaps in the valid case (and most in general), whereas the test for whether only one item in the pair is signed must be performed for each signature on the document.

2.3 Securing Empty Spaces on the Face of a Document

The requirement of Assertion 1 to sign what is seen certainly includes signing all necessary items that represent

¹¹At 120 dots per inch, a letter-sized page is 1320 dots high.

¹²In languages that flow vertically, the overlap test can be performed with a vertical sweep line and events based on sorted x coordinates.

¹³Many forms are also restricted to an 8 inch (960 dot) width for printing purposes.

form controls such as labels and text input items. However, to prevent unsigned items from being placed near signed items in positions that may alter the signed items' interpretation, certain empty spaces on the face of a form must also be protected.¹⁴ The decision of whether an empty space is allowed to contain unsigned items is document-specific and can only be made by the document author who understands the visual design. Interestingly, the decision of whether an empty space must remain empty can be considered as a question of whether or not the space can be obscured by non-emptiness. Thus, to protect an empty space, the document author can place a borderless, empty, transparent, non-interactive, non-grouping item (e.g. a label) in the space and ensure that it is included in the signature. Any unsigned item moved to the same location will cause the overlap test to report a security violation. In essence, the overlap test can be used to prevent unsigned items from appearing outside of the appropriate areas defined by the document author.

3. TESTING FOR SIGNIFICANT LAYOUT CHANGE OF SIGNED ITEMS

In accordance with the requirement of Theorem 1.3, the test described in this section is designed to prevent changes to the relative position and size of signed items in positive pixel space as well as migration of signed items to or from negative pixel space. Note that one could simply append to the signed items' serialization a string that gives the absolute coordinates of the bounding boxes of all signed items. If the absolute positions of all signed items cannot change, then certainly their positions relative to one another cannot change. Section 3.1 describes why a more flexible scheme is preferable, and Section 3.2 presents the algorithm and its proofs of correctness and of $O(n)$ running time.

3.1 Rationale for Flexible Relative Layout Protection

A scheme that satisfies the requirement of Theorem 1.3 by preventing any change of absolute coordinates of the bounding boxes of signed items after the signing event is unnecessarily restrictive. For greatest flexibility, the solution should satisfy the requirements without creating additional restrictions. Capturing the relative layout rather than the absolute layout of the signed items at the time of signing admits the ability to perform a vertical or horizontal translation of the signed items, which may be necessary to complete the work on the document. If the document author does in fact want to protect the absolute positions of relatively positioned signed items, then the first item at the root of all relative positioning dependencies can simply be given an absolute position, which is typically described in the item's content and therefore protected by the core signature validation [14].

A translation of the signed items should not, however, be permitted if a signed item changes visibility by translation to or from negative pixel space. Given a pixel space *origin* of the top-left corner of a page, *positive pixel space* extends downward and rightward. The document's vertical and horizontal scroll bars permit arbitrary growth of a form in positive pixel space, i.e. beyond the bottom and right

¹⁴Such empty spaces may be necessary to faithfully reproduce the appearance of a paper form.

margins of the form. *Negative pixel space* is above the top margin and to the left of the left margin of the document. An item is defined to be in negative pixel space if its left or top margin is in negative pixel space. This prevents items that are partially visible or invisible at the time of signing from becoming more or less visible after the signature is affixed.

3.2 Significant Layout Change Detection

A *section* is a logical collection of items on a page of a document. Typically, each page has only one section, but as an example of multiple sections, consider a form in which each signer's contributions are added consecutively, but all signatures are at the bottom of the page. Each signature would include the contributions of the signer and his predecessors in one section and the signatures of the signer and his predecessors would be in a second logical section. This allows the next signer's contribution to shift the latter section further down on the page without breaking the layout test.

To test for significant layout change in a section, an initially empty *layout descriptor string*, labelled S , is calculated for each section of signed items as follows:

1. From among the bounding boxes of all signed items in a section that are in positive pixel space, determine the least left margin L and least top margin T .
2. For each of a section's signed items in positive pixel space, add to S the signed item's identifier within the document followed by its left margin coordinate less L , top margin coordinate less T , and the width and height of the item.
3. For each of a section's signed items in negative pixel space, add to S the signed item's identifier within the document and the four absolute coordinates describing the item's bounding box.

The hash of each layout descriptor string S is added to the signed information (the message to be signed by the signature). This prevents the size of the signed information from growing linearly with the number of signed items.

After the core validation events for the digital signatures, the overlap test for each page can be immediately succeeded by the test of relative layout of signed items by simply recomputing the layout descriptor string for each section on the given page and comparing its hash for equality to the hash of the section's layout descriptor string stored in the signed information. Theorem 3.1 establishes the running time of the algorithm, and Theorem 3.2 establishes its correctness.

THEOREM 3.1. *The significant layout change detector adds a summative $O(n)$ term of work to the cost of the digital signature sign and validate events.*

PROOF. For step 1, finding L and T are computations of minima over the list of signed items, which is an $O(n)$ operation. The subsequent contribution of each signed item in step 2 or 3 takes an $O(1)$ operation to construct and append to S . Since computing the hash of S is a linear time operation on S , the claim of linear time performance is proven for the signing event. For the validation event, the same steps are performed, followed by an $O(1)$ comparison per section between the computed layout descriptor string and the hashed layout descriptor string in the signature. \square

THEOREM 3.2. *The layout test reports failure if and only if there exists a signed item I_1 that changes at least one of the following:*

1. *width or height,*
2. *location between the page's positive and negative pixel spaces*
3. *location if in the page's negative pixel space,*
4. *location relative to some other signed item I_2 if both are in positive pixel space and in the same section.*

PROOF. If a signed item I_1 changes width or height, then the width or height directly changes in the layout descriptor substring for I_1 . If a signed item I_1 changes between the page's negative and positive pixel spaces, then either the left or top margin specifier in the layout descriptor substring for I_1 will change by at least the appearance or disappearance of a negative sign. Moreover, if I_1 was in negative pixel space at the time of signing and remains in negative pixel space for the layout test, then any change to its left or top margin changes the corresponding location specifier in the layout descriptor substring for I_1 .

On the other hand, if the signed item I_1 does not experience one of the changes in conditions 1 to 3, then the width and height portions of its layout descriptor substring do not change, and its left and top margin specifiers cannot change unless I_1 is in the page's positive pixel space and was also there at the time of signing. Hence, we consider the final case in which a relative location change occurs between the signed item I_1 in the page's positive pixel space and another signed item I_2 also in the page's positive pixel space and in the same section.

If the left margin of I_1 changes by C units without a corresponding change to the item I_2 that defines L , then the left margin coordinate in the layout descriptor substring for I_1 changes by C units. Similarly, if the top margin of I_1 changes by C units without a corresponding change to the item I_2 that defines T , then the top margin coordinate in the layout descriptor substring for I_1 changes by C units. Likewise, if the left margin of the signed item I_1 that defines L changes without a corresponding change to all other signed items in the page's positive pixel space and in the same section, then the layout descriptor substrings will change for each signed item $I_2 \neq I_1$ in the same section of the page. Similarly, if the top margin of the signed item I_1 that defines T changes without a corresponding change to all other signed items in the page's positive pixel space and in the same section, then the layout descriptor substrings change for each signed item $I_2 \neq I_1$ in the same section of the page. Thus, any horizontal or vertical translation that is not uniformly applied to the left margins or top margins of all signed items in the page's positive pixel space and in the same section results in some change to the section's layout descriptor string.

On the other hand, if the signed items in a section that are in positive pixel space are subjected to a uniform horizontal translation H or vertical translation V (such that all of the items remain in positive pixel space), then the values L and T become $L + H$ and $T + V$. Thus, translations H and V are removed from the translated left and top margin coordinates of each signed item and the layout descriptor string will not differ from the original value stored in the signed information. \square

4. CONCLUSION

A document subset signature is affixed when further work must be performed on a document after the signature is affixed. Before serializing the document, a document filter is applied to describe the exact portion of the document to be omitted from the signature. In XML Signatures [14], XPath expressions [9] and set operations on subtrees of the document parse tree [6] are used to express document filters.

The W3C's XForms working group is beginning to consider the security of electronic forms, and the group lists integration with XML Signatures [14] among the highest priorities of its new standardization charter. The first section of this paper has provided proofs of necessity for additional requirements that must be considered in the standardization process in order to address the most demanding signature scenarios for electronic forms. Two security problems that can arise from the use of document subset signatures were presented. These problems can be difficult or impossible to solve using only document filtering because the information dependencies at the center of the problems result from application semantics and data that is either unavailable to or not easily characterized by a document filter.

In order to be scalable, the solutions for the two problems must permit implementations that achieve an expected $O(n)$ rating since core cryptographic signature validation is a linear time operation. This paper presented solutions that meet the required performance profile.

The first solution, the overlap test, detects forbidden overlap of the bounding boxes of user interface elements (items). The basic algorithm resembles the computational geometry algorithm for rectangle intersection, but it has been optimized to exploit special features of the problem domain to achieve expected $O(n)$ performance, which is necessary to match the performance profile of the core digital signature calculations. Moreover, practical concerns mandated several subsequent tests to determine whether a given overlap was forbidden or permitted, such as overlap below a defined tolerance, overlap for which each signature either omits or retains both items, and certain overlaps involving items that provide background effects and visual grouping within the user interface. The overlap test was also found to be useful in securing empty spaces near signed items so that the interpretation of the signed items cannot be changed by filling the empty spaces with unsigned items.

The second solution, the significant layout change test, detects whether the relative layout of signed items has changed since the signing event. For practical reasons, it is necessary to allow translations of the set of signed items along the positive x and y axes, but a change of relative position between two signed items or a translation to or from negative pixel space is forbidden. At signing time, a layout descriptor string is computed for each logical section of a page, and a representative of it is stored as part of the signed information in the digital signature. The layout descriptor string must be faithfully reproduced during layout validation. To prevent the size of the resulting digital signature from growing with the number of signed items, a hash of the layout descriptor string is used as its representative.

5. REFERENCES

[1] M. Baum and R. Schwartz. (Eds.) *Digital Signature Guidelines: Legal Infrastructure for Certification Authorities and Secure Electronic Commerce*. American

Bar Association, Section of Science and Technology, 1996. Available at: <http://www.abanet.org/scitech/ec/isc/dsgfree.html>

[2] R. Bertrand, J. Hearn and B. Lett. *The North American Pre- and Post-Processing Equipment Market: Capturing the Benefits and Avoiding the Pitfalls*. Strategic Analysis Report, Gartner Group, September 1995.

[3] B. Blair and J. Boyer. XFDL: Creating Electronic Commerce Transaction Records Using XML. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 31, pp. 1611-1622, 1999.

[4] J. Boyer. Improper Utilization of Digital Signature Technology. *RSA 2000 Conference Proceedings*, San Jose, California, January 2000.

[5] J. Boyer, T. Bray and M. Gordon (Editors). *Extensible Forms Description Language (XFDL) 4.0*. W3C Note, September 1998. Available at: <http://www.w3.org/TR/NOTE-XFDL>

[6] J. Boyer, M. Hughes and J. Reagle. *XML-Signature XPath Filter 2.0*. W3C Recommendation, November 2002. Available at: <http://www.w3.org/TR/xmlsig-filter2/>

[7] T. Bray, J. Paoli, and C.M. Sperberg-McQueen (Editors). *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C Recommendation, October 2000.

[8] R. Casonato. *Integrated Electronic Form Management Strategies*. Inside Gartner Group, February 1996.

[9] J. Clark and S. DeRose (eds.). *XML Path Language (XPath) Version 1.0*. W3C Recommendation, November 1999. Available at: <http://www.w3.org/TR/xpath>.

[10] J. Clark. *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation, November 1999. Available at: <http://www.w3.org/TR/xslt>.

[11] Cohasset Associates, Inc. *What's New: Electronic Commerce Forms: Requirements, Issues, and Solutions*. Available at: http://www.cohasset.com/main/library/coh_articles/whatnew_body_ecforms.htm

[12] M. Dubinko, J. Dietl, L. Klotz, R. Merrick and T. V. Raman (eds.). *XForms 1.0*. W3C Proposed Recommendation. August 2003. Available at: <http://www.w3.org/TR/xforms/>

[13] D. Eastlake and K. Niles. *Secure XML: The New Syntax for Signatures and Encryption*. Addison-Wesley, Boston, Mass., 2003.

[14] D. Eastlake, J. Reagle, D. Solo, M. Bartel, J. Boyer, B. Fox, B. LaMacchia and E. Simon. *XML-Signature Syntax and Processing*. W3C Recommendation, February 2002. Available at: <http://www.w3.org/TR/xmlsig-core/>

[15] J. Ferraiolo (ed.) *Scalable Vector Graphics (SVG) 1.0 Specification*. W3C Recommendation, September 2001. Available at: <http://www.w3.org/TR/SVG/>

[16] F. Preparata and M. Shamos. *Computational Geometry*. Springer-Verlag, New York, 1985.

[17] D. Raggett, A. Le Hors and I. Jacobs (eds.). *HTML 4.01 Specification*. W3C Recommendation, December 1999.