

DIGITAL SIGNATURES AND ELECTRONIC DOCUMENTS: A CAUTIONARY TALE*

*To appear, Sixth IFIP Conference on Communications and
Multimedia Security, September 2002*

K. Kain

*Dartmouth College
Hanover, New Hampshire, USA
kunal@Dartmouth.edu*

S.W. Smith

*Dartmouth College
Hanover, New Hampshire, USA
sws@cs.Dartmouth.edu*

R. Asokan

*Virginia Tech
Blacksburg, Virginia
rasokan@vt.edu*

Abstract Often, the main motivation for using PKI in business environments is to streamline workflow, by enabling humans to digitally sign electronic documents, instead of manually signing paper ones. However, this application fails if adversaries can construct electronic documents whose viewed contents can change in useful ways, without invalidating the digital signature. In this paper, we examine the space of such attacks, and describe how many popular electronic document formats and PKI packages permit them.

Keywords: PKI, digital signatures, e-commerce, e-government.

*This work was supported in part by the Mellon Foundation, by Internet2/AT&T, and by the U.S. Department of Justice, contract 2000-DT-CX-K001. The views and conclusions do not necessarily represent those of the sponsors.

1. The Problem

One of the most common uses of public-key cryptography is for *digital signatures*. If Alice performs a private-key operation on an electronic object O (usually via hash and padding functions) to yield a signature $S(O)$, then those who believe in the PKI can verify that $S(O)$ came from O via Alice’s public key—and thus conclude that Alice generated this signature.

Typically, O itself may consist of an object O_1 and a field I indicating intention, such as “Alice approves O_1 ” or “Alice witnesses that O_1 arrived by some point in time.” In the process of signing, Alice takes O_1 , adds I , and signs the result: indicating her approval or witness.

In the non-digital world, people must frequently take such action on *paper* documents: e.g., signing forms and expense sheets and contracts; recording when a bid or homework assignments was submitted. The last decade has seen a revolution: these paper documents have migrated into electronic settings; rather than dealing with a paper expense form, we deal with an Excel spreadsheet. This change in media allows a revolution in the ease and speed of creating and sharing documents, even between parties on opposite sides of the Internet.

The natural question arises: since we often need to sign paper documents, and a PKI would permit a nice way to sign electronic objects, can we compose the two, and indicate personal approval of a “virtual” paper document, by digitally signing the corresponding electronic object? The ability to do this composition is often a main motivating factor in deploying enterprise PKI (such as ACES [15]); businesses and products exist to provide exactly these services (we have surveyed some for this project [2, 3, 5, 9, 10]).

Typical electronic workflow tools replace a paper document with an electronic object O , that yields a virtual piece of paper when a party opens the object. When Alice signs an object O , she commits to the virtual piece of paper she sees when she views the object. By PKI, when Bob verifies $S(O)$, he concludes that Alice digitally signed O . By composition—if we accept that digital signatures imply workflow approval—Bob concludes that Alice approves the virtual paper document that O represents. Figure 1 sketches this scenario.

We decided to investigate whether this composition actually works. The particular angle that struck us as questionable was the implicit assumption that electronic objects generate semantically equivalent virtual documents each time they are opened. If $V_B(O)$ can be made to be substantially different from $V_A(O)$, then Bob’s conclusion about what Alice signed will differ from what Alice thought she signed.

In Section 2, we discuss scenarios in which an adversary (perhaps Alice, or perhaps the entity who gave Alice the document) can benefit by constructing such objects. In Section 3, we discuss how popular workflow formats permit the construction of such objects. In Section 4, we discuss how many common

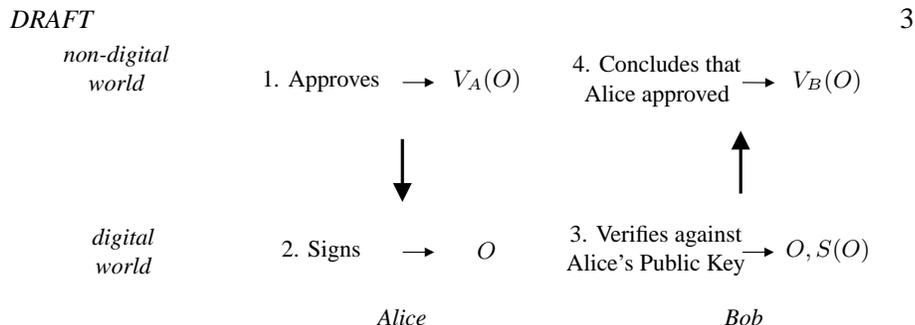


Figure 1. Current PKI/workflow integration attempts to capture the non-digital process of signing paper, by digitally signing the corresponding electronic object. To “sign” the virtual document $V_A(O)$, Alice digitally signs the corresponding electronic object O . If Bob receives O and $S(O)$ and verifies this signature, then he concludes that Alice approved the virtual document $V_B(O)$ that he sees.

ways of integrating PKI with workflow can still appear to validate signatures on such objects (although some packages resisted our attacks). In Section 5, we consider some countermeasures.

1.1. Prior Work

Our work was motivated by planned deployment of real applications that used digital signatures on electronic documents. We consequently began investigating the surprising extent to which these these document formats were malleable, and to the surprising extent to which COTS PKI packages tolerated this malleability.

However, these issues certainly have an older history. For one example, Herzberg [8] considered signing the viewing program as well as the document. For another example, we can consider Europe. The European Union’s electronic signature directive [4] made electronic signatures at least as binding as paper-based signatures. Germany actually had a digital signature act (1997) even before the EU’s Directive; in fact the directive was adjusted to conform to the German Digital signature Act [13]. Austria has also fully implemented the directive[1]; and concretized the malleability problem by specifying that only data formats may be used which have an “available specification” and which exclude “dynamic changes” or “invisibilities.”

Pordesch [11, 12] has also considered these issues.

2. Attack Scenarios and Approaches

2.1. Motivating Scenarios

When performing vulnerability analyses, if one discovers some avenue to cause the system to engage in unexpected ways, one typically encounters the

response that no one would ever do that. Consequently, we begin by pro-actively addressing this concern.

Within our university, two applications motivating campus PKI are *timestamped homework submission* and *signing of payroll and expense forms*.

- The timestamped homework application typically posits that a student Alice submits homework to an automatic system Bob that appends the current time, digitally signs the result, and forwards both on to the instructor Cathy. (One could also easily imagine a system using more advanced timestamping techniques. [7].)

Suppose student Alice could construct an electronic document O that, whenever it was viewed, first consulted a remote file or Web location, and rendered a virtual document based on the contents of that remote file R . Alice could then completely subvert the purpose of timestamping by submitting O before the assignment deadline, but continuing to work on R up to the time the project was graded. If the assignment is one where the instructor Cathy posts sample solutions before grading, Alice could guarantee herself high scores by preparing R to follow those solutions.

- In the typical processing of expense forms, a submitter Alice sends a form to an approver Bob, who then sends it to Cathy, who actually issues checks. Suppose malicious Alice has two sets of numbers: a set S_1 with illegitimate expenses that Bob would not approve but which would result in Cathy issuing a large check, and a set S_2 with smaller numbers that Bob would approve. If Alice could construct an electronic document O that displays S_2 when viewed by Bob but S_1 when viewed by Cathy, then Cathy will receive a Bob-approved expense form indicating S_1 , and Alice will a larger reimbursement than she should.

In the above two examples, Alice benefits by obtaining Bob's signature on an electronic object that, when viewed in some contexts, displays a virtual document that Bob would not have signed. Scenarios also exist where Alice could benefit by being able to retroactively change documents to which she previously committed. For example, Alice might submit a signed bid to provide (or purchase) services, and might benefit from retroactively changing the terms of that bid.

The domains of commerce, legal processes, and medical processes offer many other examples.

2.2. Taxonomy of Approaches

We now sketch a rough taxonomy of ways an adversary might construct such malleable documents.

DRAFT

5

Hidden Parameters. This avenue of attack works when the virtual document that appears when someone views an electronic object O is not completely determined by O alone, but instead depends in part on other parameters. One way to characterize attack strategies is to consider these parameters. We offer some:

- **Time.** Can the virtual document usefully change depending on the time the object is viewed?
- **Viewer Data.** Can the virtual document usefully change depending on the identity of the viewer, their machine, their operating system, or other such context data?
- **Viewer Action.** Can the virtual document usefully change depending on actions the viewer takes?
- **Remote Control.** Can the virtual document usefully change depending on the existence or contents of a file controllable by the adversary? A secondary issue here would be the proximity of this file to the viewer: a strategy that permitted the file to be an arbitrary URL would require only that the attacker control a web site, but also would require that the viewer have a good connection to the Web (in some cases, we have been able to overcome this restriction by using a 1x1-pixel image to pre-load the required document in the viewer's cache). Having the file to be closer to the viewer may require the attacker to have greater access.

Note that a Web-based remote control attack can potentially be used to mount a viewer-specific attack, if the viewer visits from a predictable host. (However, we did not try this in our tests.)

Fraudulent Content. In our attack scenarios, the adversary devises a way to usefully change the apparent content of a signed document. The question then arises of *when* the adversary must determine this alternate content. Two natural choices suggest themselves:

- **Pre-signature.** The alternate content must be fixed at the time the signature is applied.
- **Post-signature.** The alternate content may be chosen at some point after the signature has been applied.

Nature of Change. Another parameter is how the display of alternate content affects the "current" electronic document.

- **Static Content.** The most powerful attack strategy is one where viewing alternate content does not change the working object.

6

- **Dynamic Content.** A strategy that requires the working object (e.g., the Word document the viewer opened) to be modified as part of displaying alternate content can still be effective, but only when the signature is verified against the original object.
- **Dynamic Content and Signature.** It is also conceivable that an attack strategy might change signatures at the same time it changes the contents of the object, perhaps by shipping pre-established object-signature pairs.

Other Angles. We intend the above simply as an illustrative taxonomy, not as a complete one. In particular, we also want to leave open other avenues for attack. For one example:

- **Spoofed Signature.** In standard Web spoofing [6, 16, 17], the adversary uses the richness of the user interface to create the illusion of the desired result. When attacking signatures, the adversary might use a similar technique: the document might modify itself and invalidate its bona fide signature—but mimic the signature-verification user interface sufficiently well that the user is still convinced the signature is valid.

3. Workflow Formats

No one is satisfied with ASCII text anymore.

In this section, we consider various popular formats for electronic documents, and the potential to realize the above attack scenarios in these formats.

We note that this is an exploratory list, not an exhaustive one. (We just wanted to see what we could find; by no means can we assert that this is complete list of all attack strategies.)

3.1. Word

3.1.1 With Macros. Since release 6.0, Microsoft Word has permitted users to add active content to documents via *macros*. In our first attempt, we explored whether we could use Word macros to carry out a *remote control, post-signature* attack. With some trial and error, this is easily done: with the opening of the document, we associate a macro that replaces the current contents with the contents of a remote file.

```
Private Sub Document_Open()  
Set doc = Documents.Open(URL of file)  
Set rng2 = doc.Range  
Documents("signed.doc").Activate  
Set rng = ActiveDocument.Range  
rng.FormattedText = rng2.FormattedText  
doc.Activate  
ActiveDocument.Close
```

DRAFT

7

This works, but is brittle: the viewing party must either have macro security to set to low or the attacker must be a trusted macro signer; furthermore, this is a *dynamic content* attack that changes the file contents.

3.1.2 Fields. The security risks of Word macros are sufficiently well-known that many users leave their Macro security setting at “high”; mounting a signature attack via this vector is not particularly plausible—nor interesting.

Instead, we decided to look at approaches that did not use macros.

One set of promising techniques is the *Insert Field* feature in Word. From the *Insert* menu, if one selects *Field*, one is given a rich set of fields and operators with which to build active content.

For example, we can carry out a *time* attack by using the conditional *IF* operator on the *DATE* field. In the fragment below, the author revises his testimony after the 16th. (A more complex conjunction operation would give us month and year checks as well.)

```
{ IF { DATE \@ "d" \* MERGEFORMAT } < 16
  "I did not have" "I did have"}
```

Installing field code is tricky: one cannot simply “type” the code into the field box. For code such as the example, we first selected the *IF* operator from the menu, then *within* the resulting code, we inserted the *DATE* field.

The above attack has the limitation of being *pre-signature* but the advantage of being *static content*: the binary document appears to contain the entire conditional, and does not appear to change depending on the branch taken.

Fields also offer promising hooks such as *USERNAME*. However, apparently, only *DATE* and *TIME* are automatically updated upon document open; for *USERNAME* to be updated to reflect the name of the current viewer, the viewer must explicitly update fields. (although the attacker can configure the document to do this when printing).

Fields can also be updated automatically by via macros, but that reduces us to the previous case.

```
Sub UpdateAllFields()
  Dim aStory As Range
  Dim aField As Field
  For Each aStory In ActiveDocument.StoryRanges
    For Each aField In aStory.Fields
      aField.Update
    Next aField
  Next aStory
End Sub
```

(We did not examine modifying Word documents via a binary editor to see if we could cause fields to behave in an undocumented fashion.)

3.1.3 Links. Somewhat unexpectedly, Word also permits the user to insert material from remote documents by reference. To do this, the user copies

text from one Word or Excel document (we have not tried other Office formats); then, in the target document, the user selects *Edit*, then *Paste Special*, then *Paste Link*, then pastes the text in as a *link* (*unformatted text* works nicely).

This approach permits a *post-signature*, *remote control* attack. Unfortunately, this also appears to be a *dynamic content* attack—Word asks whether to save the document.

A surprising side-effect of the existence of this feature in Word is that a remote Web site can track each time one reads a document, and even plant cookies. (The University of Denver [14] has also noticed this “feature”.)

3.2. Excel

3.2.1 Macros. As with Word, Excel also has powerful Macro capabilities. Again, we decided not to explore this direction, because the risks are already well-known.

However, the plausibility of Excel macros as an attack vector might be greater than Word’s. Many organizations use macro-laden spreadsheets as standard practice. For example, universities preparing grant proposals for the US *National Science Foundation (NSF)* are required to download Excel spreadsheets with Macros. Since these spreadsheets typically get routed throughout university administrative staff, at least one large population is primed to always hit the “accept macros” button.

3.2.2 Time and Date. In Excel, the user can associate functional behavior with specific cells. This behavior has interesting potential for our purposes.

For example, an attacker can mount a *pre-signature*, *time-based* attack by selecting *Insert*, then *Function*, then building an *IF* construct using *NOW*(). A simple example:

```
IF(DAY(NOW())<16, 2000,20000)
```

Unfortunately, this is a *dynamic content* attack, since Excel appears to notice the cell is “volatile” and asks whether the document should be saved.

3.2.3 Operating System. We can also use functions to mount *viewer-data* attacks—although the most useful viewer data we could find was operating system (or perhaps file path name).

For example:

```
IF(INFO("osversion")<>"Windows (32-bit) NT 5.00",
"I love Linus","I love Bill")
```

Again, this attack is *pre-signature*, but also is *dynamic-content*.

3.2.4 Links. The same techniques of Section 3.1.3 apply to Excel as well: the attacker can copy data from a Word or Excel file under his control,

DRAFT

9

then *Paste Special* a link into an Excel cell. As before, this enables a *remote-control, post-signature, dynamic-content* attack.

Unfortunately, this appears to trigger a warning each time the file is opened. Options exist to disable this warning (*Tools*, then *options*, then *Edit*), but these appear to remain with the installation of Excel, rather than traveling with the file.

3.2.5 External Queries. Excel includes features to make explicit queries to remote files. These features enable *post-signature, remote control* attacks that have *dynamic content*.

From the *Data* menu, the attacker can select *Get External Data* and then set up a query to a remote text file. The text file should be written with tab spaces between words to specify different fields in the spreadsheet. By right-clicking on the cell and selecting *Data Range Properties*, the attacker can configure the query to update on open or even regularly (in the background).

Sometimes, this technique gives a warning and an external data toolbar pop-up.

3.3. PDF

Adobe's *Portable Document Format (PDF)* is fast becoming an alternative to Word as the *de facto* standard for electronic documents.

Various tools exist to view PDF and to convert other formats into PDF; however, using the official Acrobat 5 product, one can more directly explore nuances in creating PDF documents.

With this flexibility—and with Acrobat's use of Javascript with event-driven actions—we were able to explore some interesting avenues (although we have not been able to carry out a remote-control attack here).

3.3.1 Time. Using some Javascript functions we were able to make attacks similar to our time-based attacks on Word and Excel.

For example, one can use the *form* toolbar to create a form field, and then add Javascript code in its *calculate* field to change the value of the field according to the date.

```
var f = 9000
var g = util.printd("d", new Date())
if(g < X) f = 5000
event.value = f
```

The variable *g* here just gets the day out of the date value, and the *X* represents some day against which we want to check the value of *g*. In this type of form field, we can make it appear without a border—like an ordinary text. We can also make it “Read-only” using the *appearance* tab.

10

3.3.2 Viewer Action. Above, we made a form that appeared to be text. We can also make a form that is invisible—but which has Javascript associated with viewer actions.

For example, when the mouse moves over the form, we can trigger Javascript to change the value of another field. We click on the *form creation* tool on the tool bar and use the mouse to create a box in the document by dragging it across the screen to the size we want the form box to be. Then, we click the *mouse enters* from the *Actions* bookmark, and select *Run Javascript* from the drag-down menu. We could then use the Javascript to give some form named “danger” another value.

```
event.value = 1
```

3.4. HTML mail

As with documents, people are not satisfied with just plain ASCII mail, and instead want mail content to incorporate feature such as colors, different fonts, and pictures. The MIME standard permits email to be formatted as HTML, and consequently many popular clients (such as Yahoo Mail, Hotmail, iname.com, and Mailcity) do this.

HTML email provides a rich breeding ground for a variety of attacks. To test these techniques, one needs to discover how to convince one’s mail client to send an arbitrary HTML file as MIME mail; in Microsoft Outlook, when sending new mail, one clicks on the *view* menu item, then selects *source edit*, then selects the *HTML* tab.

3.4.1 Date. Using Javascript as a tool we carried out a variety of attacks in this area.

For example, we can change the content of the document with a change in date by using the `document.write()` method which allows us to write a new page using HTML tags. With many mail clients (including Netscape and Outlook), if the user attempts to look at the HTML source for this mail, they only see the final HTML—and not the Javascript with the dynamic content.

```
<HTML>
<HEAD>
<TITLE>HTML Mail</TITLE>
<SCRIPT language=JavaScript >
function InitForm()
{
var today = new Date()
var day = today.getDate()
if (day > 8)
{
document.write("<PRE><BR>Kunal    4000<BR>Sean    8000<BR></PRE>")
}
else {
document.write("<PRE><BR>Kunal    8000<BR>Sean    16000<BR></PRE>")
}
```

DRAFT

11

```

    }
  }
</SCRIPT>
</HEAD>
<BODY onLoad="InitForm();" >
<PRE>
</PRE>
</BODY>
</HTML>

```

We can similarly do changes based on time.

3.5. Remote Control and Viewer-Specific Attacks

Another type of attack that we can carry out is include, in the HTML, references to inline images which reside on a remote system. This technique permits post-signature remote-control attacks (since we can change the images after the fact) as well as viewer-specific attacks (since our Web server can track which user is requesting the images).

We successfully carried out this attack with an image that looked like text.

```

<HTML>
<HEAD>
<TITLE>HTML Mail</TITLE>
<BODY>
<PRE>
<IMG name=thumbnail src="http://www.cs.dartmouth.edu/~kunal/trial1.jpg">
</PRE>
</BODY>
</HTML>

```

As noted, it is possible to get all kinds of information from already existing methods from the browser type to the operating system being used.

We can do this at the client-side by using Javascript:

```

<HTML>
<HEAD>
<TITLE>HTML Mail</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function InitForm()
{
if (navigator.appName == "Microsoft Internet Explorer")
{
document.write("<PRE><BR>Kunal      4000<BR>Sean      10000<BR></PRE>")
}
else {
document.write("<PRE><BR>Kunal      8000<BR>Sean      16000<BR></PRE>")
}
}
</SCRIPT>

```

12

```

</HEAD>
<BODY onLoad="InitForm();" >
<PRE>
</PRE>
</BODY>
</HTML>

```

Using Microsoft Outlook, which uses the Microsoft Internet Explorer to view HTML mail, we were able to successfully carry out this test.

4. Methods of PKI/Workflow Integration

We now consider some sample approaches used in the field for signing virtual documents.

4.1. External PKI

One approach is to use a PKI package that is completely oblivious to documents it is signing. For example, one might use PGP to sign and verify documents sent as email attachments. For another example, we also obtained a certificate from Digital Signature Trust [2] and used their *CertainSend* application to sign, transmit, and verify signatures on electronic documents. With these approaches, the above attack strategies on Word, Excel, and PDF work.

We also worked with S/MIME and found it vulnerable to the HTML email attacks. Experimenting with Outlook and using its built-in signature and encryption features, we were able to carry out the attacks proposed in Section 3.4 successfully.

4.2. Assured Office

Another approach is to use a PKI package that is explicitly integrated with the document software. For example, the *Assured Office* product from E-Lock Technologies [5] adds sign and verify buttons in one's Office installation, and uses a key pair resident in the user's browser to sign documents.

Table 1. A summary of the attacks we tried

Types of attacks	Data Format Used
Time/Date-Based Attacks	Microsoft Word/Excel, Adobe Acrobat, HTML mail
Macro-Based Attacks	Microsoft Word
Linked file Attacks	Microsoft Word/Excel, HTML mail
Platform-Based Attacks	Microsoft Word/Excel, HTML mail
Event-Based Attacks	Adobe Acrobat

With this approach, all of the Word and Excel attacks we outline above still work. Even the dynamic-content attacks work, because the signature is verified at document open, before the changes have been made.

(E-Lock has recently been acquired by Lexign [9], and the Assured Office product has been renamed ProSigner.)

4.3. Acrobat Signed PDF

Adobe Acrobat has two types of signature features built in.

4.3.1 Visible Signatures. In Adobe *visible signatures*, the document's signature is visible as an image that indicates whether the signature has been verified and whether the document was modified since. Our time-based attack of Section 3.3.1 still worked (although some versions with updates every second seemed to change the document before the signature was verified).

In theory, we could also use the techniques of Section 3.3.2 to add mouse-over action to the digital signature field—for example, to change a global value when the user verifies the signature, so interesting things can happen after the signature is verified.

The visible signature approach is also potentially vulnerable to plain old spoofing: including an image that looks just like a valid, verified signature.

4.3.2 Invisible Signatures. In Acrobat's *invisible signatures*, the signature is applied as an invisible tag. This technique extremely well against the attacks we have proposed, as the *image* of the document can easily be matched after a change in the file content with the previous image and the changes can be easily noticed.

4.4. Utimaco

Another product we have recently discovered is the *SafeGuard Sign&Crypt for Office* from Utimaco Safeware [3], in Germany. Although we have not been able to obtain sample tools, we were pleased to notice that their online documentation indicated they were concerned about “invisible dynamic content,” and the screenshots indicate they appear to sign and verify TIF images of documents. We speculate that our attack strategies would not be effective here.

4.5. Silanis

Silanis [10] of Canada makes a *ApproveIt* package for Word and Excel documents. In our tests, ApproveIt seems to be aware of most of our attempts to change document contents, and blocks the field contents to be updated. However, our tests also showed that ApproveIt still permits macro-based attacks.

14

5. Conclusions and Countermeasures

We consider several general approaches for potential countermeasures:

- **Inert Documents** One approach is to carefully design a document format that is inert: has no dynamic content. (This appears to be the approach taken by Utimaco.) Given the surprising richness of document formats, we would hesitate to certify any given one as inert. We also wonder whether users would accept inert documents in their workflow process.
- **Application Awareness** Another approach is to make the digital signature tools highly application aware, and refuse to verify (or perhaps even sign) documents that had malleable content. Acrobat Invisible Signatures appear to move in this direction. The fact that Excel even flags certain of our tricks as “volatile” and notices the document changes suggest an avenue to correct those issues.

However, this approach would complicate the acceptable API that PKI tools should offer to applications.

- **Identify and Sign Parameters** Herzberg [8] once proposed signing both a document and the program that views the document. One might extend that to explicitly identify and sign all hidden parameters; however, it has been observed that this may not be a feasible solution.
- **Verification Cleanrooms** Alternatively, one might design “safe” places, free of predictable influences, where a document might be verified. For example, our remote-control Web attacks can be detected if the verifier has a slow or non-existent Web connection. (Of course, this notion of moving *verification* to a trusted safe place runs counter to the standard intuition of moving *signatures* there.)

We note that some products we tested resisted many of our attacks. and that the US appears to lag behind Europe with regard to laws and standards in this area. We would urge application deployers to carefully examine their tools.

What additional lessons do we draw from this?

First, this work offers more data points in support of standard security ca-nards:

- The *composition* of apparently reasonable systems in not necessarily secure. What other things are dangerous to simply sign?
- As we saw before with web spoofing, the *surprising functionality and interoperability* of common desktop tools yields many opportunities for malicious behavior.

DRAFT

15

Second, by a simple exploration, we've stopped our university from deploying a fatally flawed signature/workflow integration; we offer this paper as a caution for others considering digital signatures on paperless transactions.

Acknowledgments

We thank our colleagues in the Dartmouth PKI Lab for the support, and the anonymous referees for their helpful comments.

References

- [1] C. Brenn. *Summary of the Austrian Law on Electronic Singatures*. <http://rechten.kub.nl/simone/brenn.htm>
- [2] Digital Signature Trust. *CertainSend Security: A Brief Technical Overview*. http://www.trustdst.com/prod_serv/certainseend/tech_overview.html
- [3] D. De Maeyer. *Interoperability at Utimaco Safeware: Digital Transaction Security*. http://www.utimaco.de/eng/content_pdf/pkic.pdf
- [4] *DIRECTIVE 1999/93/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 13 December 1999 on a Community framework for electronic signatures*. http://europa.eu.int/ISPO/ecommerce/legal/documents/1999_93/1999_93_en.pdf
- [5] E-Lock Technologies. *E-Lock Technologies Assured Office*. http://www.elock.com/pdf/ao_entrust.pdf
- [6] E. Felten, D. Balfanz, D. Dean, and D. Wallach. "Web Spoofing: An Internet Con Game." *20th National Information Systems Security Conference*. 1996.
- [7] S. Haber and W. Stornetta. "How to Time-Stamp a Digital Document." *Journal of Cryptology*. 2:99-111. 1991.
- [8] A. Herzberg. Personal communication.
- [9] Lexign Incorporated. *The Lexign Suite*. http://www.lexign.com/resources/white_papers.htm
- [10] D. McKibben. *Silanis Technology: Signature Technology for E-business*. http://www.silanis.com/download/whitepapers/silanis_gartner.pdf
- [11] U. Pordesch. "Der fehlende Nachweis der Präsentation signierter Daten." *DuD—Datenschutz und Datensicherheit*. 2/2000.
- [12] U. Pordesch and A. Berger. "Context-Sensitive Verification of the Validity of Digital Signatures." *Multilateral Security for Global Communication* (Müller, Rannenberg, eds.). Addison-Wesley-Longman, 1999.
- [13] A. Rossnagel. "Digital Signature Regulation and European Trends." <http://www.emr-sb.de/news/DSregulation.PDF>
- [14] R.M. Smith. "Distributing Word Documents with a locating beacon." *SecuriTeam*. August 2000. <http://www.securiteam.com/securitynews/5CP13002AA.html>
- [15] U.S. General Services Administration. *Access Certificates for Electronic Services*. <http://www.gsa.gov/aces/>
- [16] Z. Ye, S.W. Smith. "Trusted Paths for Browsers." *USENIX Security*. 2002.
- [17] E. Ye, Y. Yuan, S.W. Smith. *Web Spoofing Revisited: SSL and Beyond*. Technical Report TR2002-417, Department of Computer Science, Dartmouth College. February 2002.