# The Dalì Attack on Digital Signature[1]

Francesco Buccafurri, Gianluca Caminiti and Gianluca Lax

DIMET dept., Università "Mediterranea" di Reggio Calabria,
via Graziella, loc. Feo di Vito, 89122 Reggio Calabria, Italy
{*bucca,gianluca.caminiti,lax*}@*unirc.it*

***Abstract*:**

The problem of ambiguous presentation of electronic documents has been deeply investigated in the recent literature mainly in the context of digital signature. Indeed, despite the intended goal of digital signature to guarantee the integrity of any signed document, the above problem demonstrates that the visualization of its content might vary, depending on the context. The main source of ambiguity known in the literature is the feature of many document formats to have a dynamic presentation depending on the execution of some embedded instruction. This is for example the case of PDF files which may incorporate java scripts. A similar problem may occur whenever a document can import external fonts. It is widely accepted that some formats like image (bitmap, tiff, etc.) and plain text (beside some specific format like PDF/A) are extremely safe from this point of view, since documents in these formats cannot be dynamic. As a consequence they are strongly recommended by technical rules of most countries for documents being signed in case a high level of trust is required. In this paper we present a new source of ambiguity of electronic documents which may regard also image files, allowing us to implement a new type of attack on digital signature aimed to obtain signed documents with potential (legal) effects different from those desired by the signer. The paper proves the attack by example and gives a possible way to contrast it.

***Keywords*:** digital signature, ambiguous document presentation, non-static visualization, cryptographic message, PKCS#7.

## 1. Introduction

Digital signature is the key issue of a number of innovative processes involving different components of the economic-social-administrative system. In particular, e-government activities should receive from digital signature a strong hint to enlarge significantly their action and their effectiveness. However, the basic property digital signature has to satisfy is that, at least as handwritten signature, it is a non-repudiable proof of both the identity of the signer of an electronic document and the statement of what such a document represents. As a consequence, every form of vulnerability should be carefully considered in order to understand whether digital signature may represent for electronic documents what handmade signature represents for traditional ones. The most critical point of the digital signature protocol is the secretness of the private key. This should be guaranteed against even sophisticated attacks, since compromising it could have disastrous consequences. The usage of enough secure smart cards (the EU law fixes to the standard ITSEC E3-high [16] the security lower bound) is a reasonable measure solving in practice the above problem. Indeed, a smart card can be considered a trusted platform even because it is not realistic to imagine that external attacks might have success. Unfortunately, digital signature is not free from serious weaknesses.

The most known weakness is strictly related to the fact that a smart card is a *handicapped* computer [36], since it misses I/O devices. As a consequence, the overall digital signature generation process cannot be considered trusted in general, since the PC, which the smart card is (necessarily) interfaced to, used to generate the digest of the document to sign, is potentially untrusted. The concrete risk is that, eventually, the PC can obtain a signature from the smart card on an arbitrarily chosen document different from the one displayed on the screen and actually chosen by the user. Clearly the user might not be aware about the existence of this signed document, so that the above problem can be considered really very severe. According to Rivest [32] there is an intrinsic contrast between having a secure device and having a 'reasonable customizable user interface' that supports downloading of applications. In other words, one could think of a very secure digital signature application running on a stand-alone (portable) computer not allowing us to run other software (i.e., a closed machine). Otherwise, in a more realistic case, the digital signature process remains inherently insecure, since PCs cannot be considered trusted platforms. Rivest [32] suggests that digital signature should not be considered as a non-repudiable proof, but simply as a plausible evidence. Thus users should have well-defined possibilities to repudiate such signatures. The problem, well known in the literature [1], [21], [39] is thus very hard, and does not admit a full solution whenever the PC is involved in the signature generation process. However, heuristic solutions aimed to mitigate it have been recently proposed [5], [7], [8], [9], [28], [29], [37].

Another well-known weakness is related to the possibility of documents to embed macro-instructions or executable code (for example, think about macros of Word documents or Javascript code of PDF documents). The problem is that a document containing instructions is not static, in the sense that the visualization of its content might vary as the variables, which these instructions exploit, change. For example, suppose that a contract includes an amount that is

[1] An abridged version of this paper appeared in F. Buccafurri, G. Caminiti, and G. Lax, "Signing the Document Content is not Enough: A new Attack on Digital Signature" in Proc. of the IEEE International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2008), August 4-6, 2008, Ostrava, Czech Republic, pp. 520-525, IEEE Press. The reader may find some more information about this result as well as a demonstration of the attack at http://www.unirc.it/firma.

Received August 20, 2008.

displayed as a result of a macro-instruction that is conditioned by the system date, in such a way that, after a given date, the amount is changed. Hopefully, digital signature should be able to avoid the modification of what a document shows to the user, in order to guarantee the information integrity not only in technical terms, but also from the perspective of the effects that the bits composing digital documents produce. In the above case, for example, clearly the bits of the digital contract do not vary, but their effect, in terms of knowledge they represent, does. Unfortunately, digital signature is not able to eliminate this drawback, since it is obtained from just the bits composing the document by transforming it, first by a cryptographic hash function and then by an asymmetric cryptographic algorithm (typically RSA [33]). As a consequence, digital signature is not able to detect the dynamic behavior of the document, and thus its dangerously dynamic legal effect.

This vulnerability is well-known [26] and the general way to contrast it is either trivially to force the user to check the document before signing, assuming that he is aware about the tools able to detect and remove possible dangerous instructions in the document, or to automatically check the document by a parser in order to remove dynamic contents.

A different source of ambiguities was discussed in [25], where the authors show how font substitution can be used to display the same digital document with different meanings on different computers.

All the above vulnerabilities, based on the ambiguous presentation of the document, can be fully contrasted if we restrict the permitted document formats to those not supporting the inclusion of instructions and external fonts, like plain text, image formats, PDF/A [31], etc. Technical rules typically take into account this problem, stating that the signature has not probative value when applied to documents embedding instructions able to modify what they represent or limiting permitted formats (see for example the Italian law [11], [34]).

We present in this paper a new result about a very insidious attack, never documented neither in the scientific literature nor in technical/legal/practitioner environments[2], whose effects are the same as the inclusion of macro-instructions or scripts in digital documents, but operating without the insertion in the tampered document of such components, thus not covered by the cases considered by law provisions, and possibly applicable also to those formats (like bitmap, tiff, pdf with no javascripts) considered extremely safe. We can folksy call this attack *Dalì attack*, from the name of the famous painter Salvator Dalì. The idea of this name is not from the authors of this paper, but from the journalist Luca dello Iacovo, who has written a nice article about this research on a national weekly magazine, very famous in Italy (Panorama, edited by Mondadori). The journalist relates the attack to some paints of the famous painter like *The Image Disappears* (1938), where a somewhat mysterious image of a bearded man (Dalì himself) and a scene with a woman appear: Dalì's moustache is her arm, his eye is her head and his beard is her skirt.

The attack is in fact based on the capability of a file of having a static polymorphic behavior. Thus a file that includes at the same time two different contents, with different encodes, each enabled by the application suitable for the respective format.

The contribution of the paper is thus simple yet net and unquestionably relevant from a practical point of view, since (1) the attack here presented succeeds against the technical infrastructure currently used and widely accepted both from law provisions and from industry standards, and, (2) it jeopardizes a trust mechanism used in many real contexts.

A witness of the above argumentation is that in [12] the Italian National Agency for Digital Administration (CNIPA) [10] has considered the results presented in this paper very significant, claiming the intention of addressing the problem here discussed in the preparation of the revised technical rules also by submitting our results to the Forum of European Supervisory Authorities for Electronic Signatures [19], which CNIPA is member of.

The structure of the paper is the following. In the next section how the digital signature mechanism proceeds is presented. The attack is described in Section 3. Section 4 describes how the problem can be solved. Finally, in Section 5 the conclusions are drawn.

## 2. Digital Signature

This paper refers to *strong* digital signature, which is digital signature based on both asymmetric cryptographic techniques and the usage of a secure external device (like a smart card or an USB token) for the generation process. In this section how the mechanism proceeds is briefly recalled, without going in depth about cryptographic aspects that are outside the scope of this work. These preliminary notions represents the background necessary to the reader to understand technical features of the attack.

The first step of the signature generation process is the computation, on the document to sign, of a cryptographic hash function, like SHA-1 [30] or RIPEMD-160 [14]. The result is called *digest* (typically 160 bits wide) of the document. The properties of the hash function guarantee that the digest can *substitute* the original document in the signature generation process since the probability of having two distinct documents producing the same digest is negligible. Consequently, the problem of finding a document colliding on a digest of another distinct document is unfeasible, so that an attacker cannot corrupt a signed document without the signature detects it. The digest is computed on the PC by the signature software (typically supplied by the certification authority) and sent to the smart

[2] We observe that a somewhat related (one-page) article on this attack was actually published in the national Czech journal CryptoWorld, Vol. 5, p. 2, May 15, 2008, authored by Peter Rybar. The paper, written in Czech and titled "Príklad útoku na podpisovaný dokument, ktorého typ nie je chránený samotným podpisom", very synthetically describes some issue related to our attack, and was published about 1 month later than our submission to the IEEE International Conference on the Applications of Digital Information and Web Technologies (Czech Republic) and our (direct and indirect) notification of our research (and our proof of concept) to governmental organizations, as witnessed by [12].

card embedding the private key of an asymmetric cryptographic cipher, typically RSA. The smart card is then enabled by the user (typically by inserting a secret PIN) to encrypt the digest by RSA with the private key, thus producing the digital signature. It is finally sent by the smart card to the signature software running on the PC in order to produce the *cryptographic message* (typically in *PKCS#7* [27] format[3]). The robustness of RSA [13] (used with enough large keys, typically 1024 bits) and the security used to manage the private key, allow us to give the so-obtained digital signature the power of non-repudiable proof of both the identity (guaranteed by a public-key X.509 certificate [23] granted by a trusted certification authority – included into the PKCS#7 message) of the provenance of the signed document and the statement of what the document itself represents. PKCS#7 is a standard defined by RSA describing a general syntax for data that may have cryptography applied to it, such as digital signature and digital envelope data. PKCS#7 and X.509 guarantee the interoperability of software for verifying signed documents. Indeed, the verification of a document $D$ is done by (1) re-computing the digest $I$ of the document $D$ using the same hash function as exploited in the signature generation process (this information is included in the PKCS#7 message), (2) computing $J$ as the result of the decryption of the signature $F$ done by means of the same algorithm as the generation step (as indicated in the PKCS#7 message) with the public key of the subscriber (included in the X.509 certificate, which is another component of the PKCS#7 message), and (3) checking that the decrypted digest $J$ coincides with the computed digest $I$. Clearly, the complete verification has to check both validity, trustworthiness and non-revocation of the certificate, but we do not focus on this step since it is not involved in the attack here presented.

## 3. The Attack

Whenever a user applies a digital signature to a document, he is aware about the document meaning because he sees the document as it is shown (typically on the screen of the PC where he applies the signature). Clearly, digital signature operates over the sequence of bits (i.e., the file) which composes the document being digitally signed. However, the meaning of the document depends on the way the document is shown to the user and thus on the software used to decode it. In other words, we can say that whereas traditional documents satisfy the nice property of direct *observability*, since they can be interpreted by humans using their senses (viewing and touching the document) mediated only by their capability of understanding the information contained in the document, digital documents are not similarly observable, in the sense that their bits become meaningful to humans only when correctly interpreted by an application and presented for instance through a computer screen. This is in fact a direct consequence of the immateriality of electronic documents, on the one hand, and of its machine-level essence, on the other hand, not allowing us to directly observe and understand their content. The nice consequence is that digital signature may be both immaterial and support-independent, like the document content itself, so that it has to be necessarily linked to the bits composing the document, apparently overcoming the weakness of handwritten signature as a proof of integrity. Even though this is technically true, as far as the detection of bit-level modification of signed documents is concerned, it is more weakly satisfied whenever the integrity of the information presented to humans is considered. Indeed, we cannot exclude the possibility that the bits composing the document produce ambiguous tangible contents. This is actually the source of the vulnerability which our attack relies on, as it can be understood from the next description.

Consider two different file formats, denoted by $A$ and $B$, such that both:

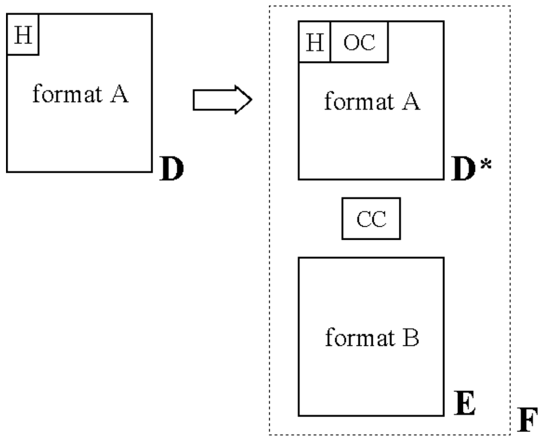
(1) $A$ is recognized by a distinguished file header, say $H$,

(2) $A$ includes an end-of-content mechanism allowing the viewer to detect the portion of the file being processed in order to display the content. For the sake of simplicity, in the sequel of the section, we refer to the existence of an end-of-content command trivially implementing the above mechanism. However, in real cases, this is not the only possible case, since the content portion could be identified through some suitable meta-data.

(3) $B$ does not require the presence of any header at the beginning of the file, and

(4) $B$ permits user comments that are skipped by any viewer of $B$.

Now we denote by $D$ the sequence of bits of a given file being digitally signed, assuming that such a file is compliant with the format $A$. The attacker must suitably incorporate in $D$ (for instance, by modifying some bytes using an hexadecimal editor) an opening comment command (denoted by $OC$) compliant with the format $B$, placing it just after the file header $H$. Call $D^*$ the sequence of bits so obtained. Then, the attacker creates a file $E$, compliant with the format $B$. Finally, he juxtaposes the bits $D^*$, the closing comment $B$-command (denoted by $CC$), and the bits $E$ to the file, obtaining thus the polymorphic file $F$. Figure 1 depicts the construction of the file $F$.



[3] Even though a number of different signature formats exist, we refer throughout this paper only to PKCS#7-signature, which is a widely used format.

**Figure 1.** The bitmap image resulting from the scan

Why the file *F* obtained as a concatenation of *D**, *CC,* and *E* is actually polymorphic?

Let us denote by $C_F(A)$ and $C_F(B)$ the presentation of the file *F* displayed by the viewers associated to the formats *A* and *B*, respectively.

Observe that $C_F(B)$ will be the result of the interpretation of just the bits of *F* compliant with the format *B*, that is *E,* since the other bits *D** plus the closing comment command just preceding the bits *E*, are skipped by the *B*-viewer, since they are interpreted as a comment in that format (we assume that the few bytes of the header *H*, which precede the comment, produce a negligible distortion of the presentation of *E*).

What about the presentation of *F* produced by the *A*-viewer?

The A-viewer will not take into account both *D*,* the closing comment *B*-command, and the portion *E* of the file *F*, since such bits are appended after the end-of-content *A*-command. Thus, $C_F(A)$ will be the result of the interpretation compliant with the format *A* of the sequence *D**. Observe that *D** is a slight modification of the original sequence *D*, since it incorporates the opening comment command (again we assume that the format *A* is such that the few bytes differentiating *D** from *D* produces a negligible distortion of the presentation of *D* – actually, we will see that in the real cases this distortion is null, since the slight modification of *D* might involve bits encoding some meta-data).

Therefore, the result is that (modulo some slight approximation), the *A*-viewer will present to the user the information encoded into *D* (in the *A* encoding), whereas the *B*-viewer will present to the user the information encoded into *E* (in the *B* encoding). In this sense, the file *F* is inherently polymorphic, and its ambiguity is activated by the switching between the utilized viewer. Observe that in most Operating Systems (like Microsoft Windows, Linux/KDE, FreeBSD/KDE, MacOs X, etc.), the viewer type is in fact established by the file extension. Thus, in these cases, the ambiguity is activated just by suitably modifying the file extension. To be more concrete, in any of the mentioned Operating Systems, if the format *A* of the scheme above is bitmap, and the format *B* is HTML, then the polymorphic file *F* will be displayed as *D* if its name has extension `.bmp`, as *E* if its name has extension `.htm`.

We have to spend some words about the concrete formats that can match *A* and *B* in the scheme above. In the previous example we have mentioned the formats bitmap and HTML. As we will see in the next sections, this choice reflects one of the possible concrete implementations of the attack schematized above. However, whereas it is feasible to find a number of formats satisfying the properties stated before for the format *A*, we argue that the only (widely) known format holding the properties required for *B* is HTML.

What happen when a user signs the polymorphic file *F*? Clearly digital signature is not able to detect these ambiguous nature of *F*, so that even though it guarantees integrity of *F* intended as sequence of bits, it is not able to inhibit its polymorphic behavior. From a practical point of view the above argumentation is definitely more meaningful whenever the extension indicates the content type (like for those Operating Systems mentioned above).

Indeed, suppose that in any of these Operating Systems the name of *F* is `contract.aaa`, where the extension `aaa` is associated to the format `A`. The content presented to the user is that displayed by the *A*-viewer, that is *D*. After the application of the digital signature, the file `contract.aaa` is included into the PKCS#7-compliant cryptographic message, which is a file named `contract.aaa.p7m`, because the digital signature software adds the further extension `.p7m` to the original document filename. Now, if the user extracts the document from the cryptographic message, the original filename is restored by discarding the previously added extension (`.p7m`). If the information about the file type is not stored inside the cryptographic message, then the verification software will be vulnerable w.r.t. the following treat: In case the file `contract.aaa.p7m` is renamed (either by mistake or maliciously) to `contract.bbb.p7m` (where `bbb` is the extension associated to the format *B*) the digital signature verification process still succeeds on it, but the extracted document will be named `contract.bbb`. Consequently, the content presented to the user (by the signature verification software) is that displayed by the *B*-viewer, that is *E*. Hence, the user has signed the content *D*, but the receiver will read the content *E*.

It is worth noting that the scheme of this attack could be weakly related to steganography techniques [38]), only concerning the feature of hiding a content into another content. However, the main properties required for a steganography technique are not satisfied by our files. Among others, observe that contents in our case are hidden only from the presentation point of view, but their presence results evident when a simple file analysis is done.

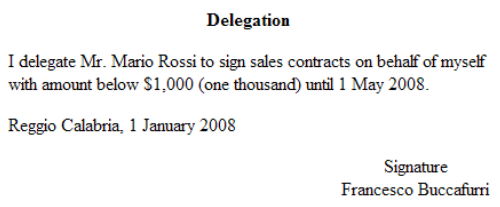
Now we show how we have applied the attack to concrete formats. In particular, we have considered (for the format *A*) `.bmp`, `.tif`, and `.pdf`. Actually, we have proven that the attack can be done also on other formats like `.ps`, `.rtf`, `.doc`, but for the sake of presentation we did not present here the description of these cases. We start by a leading example, introducing a typical scenario and choosing `.bmp` as the format *A* and Windows as Operating System (recall that the format *B* is always HTML).

## 3.1 Attack on BMP

Prof. Buccafurri wants to delegate his assistant Mr. Mario Rossi to sign sales contracts on behalf of himself with amount below $1,000. Prof. Buccafurri commissions Mr. Rossi to produce the electronic document to be signed.

In order to avoid the possible insertion of macro-instructions or executable code into the document, Prof. Buccafurri asks his assistant to obtain the document as a scan of a printed document (we consider this case as the least advantageous case w.r.t. the attempts of generating documents having non-static visualization).

Mr. Rossi (who is actually a thief), in order to carry out the attack, behaves as follows: He generates a bitmap file named `delegation.bmp` representing the image *I* resulting from the scan (see Figure 2).



**Delegation**

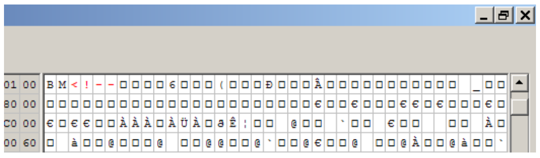
I delegate Mr. Mario Rossi to sign sales contracts on behalf of myself with amount below $1,000 (one thousand) until 1 May 2008.

Reggio Calabria, 1 January 2008

Signature
Francesco Buccafurri

**Figure 2.** The bitmap image resulting from the scan

Then, by using an hexadecimal editor, understanding the bitmap format [6], he modifies some bytes of the `.bmp` file by inserting an opening HTML comment (`<!--`) just after the two first bytes of the file (encoding the bitmap format), as reported in Figure 3. Note that the file portion following the opening comment is skipped by any HTML interpreter.



```
01 00  BM<!--□□□□6□□□(□□□Đ□□□Â□□□□□□□□□□□ _□□
80 00  □□□□□□□□□□□□□□□□□□□□□□€□□€□□□€€□€□□□€□
C0 00  €□€€□□ÀÀÀ□ÀÜÀ□∂Ê¦□□ @□□ `□□ €□□  □□ À□
00 60  □ à□□@□□□@ □□@@□□@`□□@€□□@ □□@À□□@à□□`
```

**Figure 3.** Bitmap tampering

Afterwards, he creates a suitable HTML file including a given text *T*, allowing the visualization of the desired (malicious) content. Such a HTML file begins with a closing HTML comment (`-->`). For the sake of presentation, the HTML code is only sketched (see Figure 4).

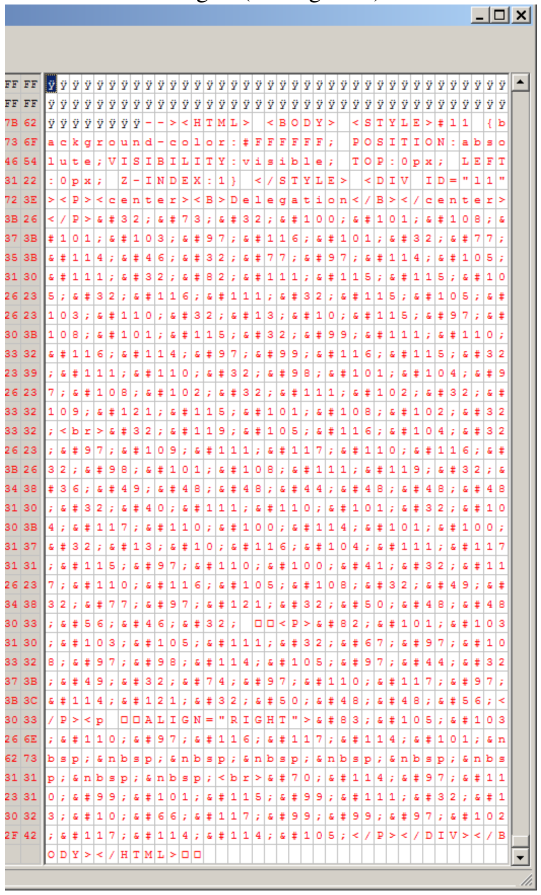
```
--><HTML><BODY><STYLE>#l1
{background-color:#FFFFFF;
POSITION:absolute;
VISIBILITY:visible; TOP:0px;
LEFT:0px; Z-INDEX:1}</STYLE><DIV
ID="l1"><table border="2" width="70%"
bgcolor="#C0C0C0" style="border-
collapse: collapse"
id="table3"><tr><td><i>Il
sottoscritto Tizio, delega Caio a
sottoscrivere contratti di acquisto
in sua vece per un valore non
superiore a <u><b>100.000</b></u>
Euro</i><p align="right"><i>In
Fede&nbsp;&nbsp;&nbsp;&nbsp;<br>Tizio
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
</i></td></tr></table></DIV></BODY></
HTML>
```

**Figure 4.** The HTML code containing the malicious content

At this point, he appends this HTML file to the previously tampered picture file. Note that, in order to contrast the possibility that the victim could detect the attack by checking the HTML source, the text *T* inside the HTML code can be obscured by using escape sequences (Figure 5).

The resulting file is correctly opened by a bitmap-viewer that shows the original image *I* (recall Figure 2). Indeed, the insertion of the opening HTML comment (`<!--`) modifies only bits encoding (redundant) meta-data, not the image *I*.

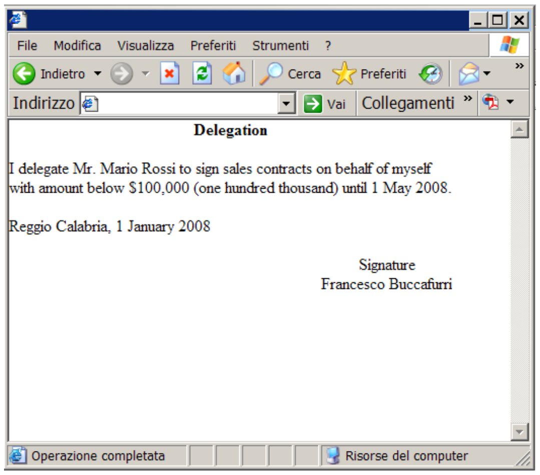
Observe that the document so created by Mr. Rossi is polymorphic. Indeed if the filename extension is changed from `.bmp` to `.htm`, the file will be opened by the associated application (the HTML browser) and shows the text *T* instead of the image *I* (see Figure 6).



```
FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
7B 62  ÿÿÿÿÿÿÿÿ--><HTML> <BODY> <STYLE>#l1 {b
73 6F  ackground-color:#FFFFFF; POSITION:abso
46 54  lute;VISIBILITY:visible; TOP:0px; LEFT
31 22  :0px; Z-INDEX:1} </STYLE> <DIV ID="l1"
72 3E  ><P><center><B>Delegation</B></center>
3B 26  </P>&#32;&#73;&#32;&#100;&#101;&#108;&
37 3B  #101;&#103;&#97;&#116;&#101;&#32;&#77;
35 3B  &#114;&#46;&#32;&#77;&#97;&#114;&#105;
31 30  &#111;&#32;&#82;&#111;&#115;&#115;&#10
26 23  5;&#32;&#116;&#111;&#32;&#115;&#105;&#
26 23  103;&#110;&#32;&#13;&#10;&#115;&#97;&#
30 3B  108;&#101;&#115;&#32;&#99;&#111;&#110;
33 32  &#116;&#114;&#97;&#99;&#116;&#115;&#32
23 39  ;&#111;&#110;&#32;&#98;&#101;&#104;&#9
26 23  7;&#108;&#102;&#32;&#111;&#102;&#32;&#
33 32  109;&#121;&#115;&#101;&#108;&#102;&#32
33 32  ;<br>&#32;&#119;&#105;&#116;&#104;&#32
26 23  ;&#97;&#109;&#111;&#117;&#110;&#116;&#
3B 26  32;&#98;&#101;&#108;&#111;&#119;&#32;&
34 38  #36;&#49;&#48;&#48;&#44;&#48;&#48;&#48
31 30  ;&#32;&#40;&#111;&#110;&#101;&#32;&#10
30 3B  4;&#117;&#110;&#100;&#114;&#101;&#100;
31 37  &#32;&#13;&#10;&#116;&#104;&#111;&#117
31 31  ;&#115;&#97;&#110;&#100;&#41;&#32;&#11
26 23  7;&#110;&#116;&#105;&#108;&#32;&#49;&#
34 38  32;&#77;&#97;&#121;&#32;&#50;&#48;&#48
30 33  ;&#56;&#46;&#32; □□<P>&#82;&#101;&#103
31 30  ;&#103;&#105;&#111;&#32;&#67;&#97;&#10
33 32  8;&#97;&#98;&#114;&#105;&#97;&#44;&#32
37 3B  ;&#49;&#32;&#74;&#97;&#110;&#117;&#97;
3B 3C  &#114;&#121;&#32;&#50;&#48;&#48;&#56;<
30 33  /P><p □□ALIGN="RIGHT">&#83;&#105;&#103
26 6E  ;&#110;&#97;&#116;&#117;&#114;&#101;&n
62 73  bsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbs
31 31  p;&nbsp;&nbsp;<br>&#70;&#114;&#97;&#11
23 31  0;&#99;&#101;&#115;&#99;&#111;&#32;&#1
30 32  3;&#10;&#66;&#117;&#99;&#99;&#97;&#102
2F 42  ;&#117;&#114;&#114;&#105;</P></DIV></B
       ODY></HTML>□□
```

**Figure 5.** Appending the HTML code to the bitmap file

Afterwards, the attacker sends the document `delegation.bmp` to Prof. Buccafurri in order to be signed.



**Delegation**

I delegate Mr. Mario Rossi to sign sales contracts on behalf of myself with amount below $100,000 (one hundred thousand) until 1 May 2008.

Reggio Calabria, 1 January 2008

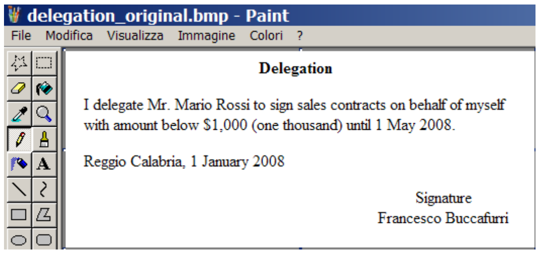
Signature
Francesco Buccafurri

Operazione completata
Risorse del computer

**Figure 6.** The result of opening the file `delegation.htm`

In Figure 7, the file `delegation.bmp` is shown by the corresponding application software (i.e. Microsoft Paint). Prof. Buccafurri signs the file `delegation.bmp` thus producing the cryptographic message in PKCS#7 format, whose filename is `delegation.bmp.p7m`. Since it has been correctly generated and no alteration has been done, the signature verification of this document clearly succeeds.

Now, the assistant completes the attack simply by changing the filename of the cryptographic message from `delegation.bmp.p7m` to `delegation.htm.p7m`. Since the signature verification is done only on the basis of the bit stream included in the PKCS#7 message, which does not contain the document filename, its change does not affect the result of the signature verification. In fact, the execution of the verification software on the renamed file succeeds, as shown in Figure 8, reporting a snapshot of the verification procedure.


delegation_original.bmp - Paint
File Modifica Visualizza Immagine Colori ?

**Delegation**

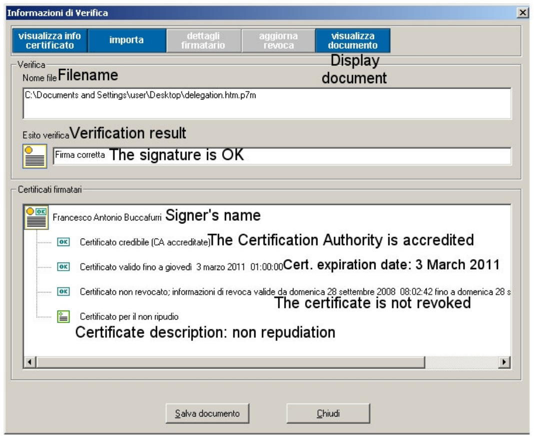
I delegate Mr. Mario Rossi to sign sales contracts on behalf of myself with amount below $1,000 (one thousand) until 1 May 2008.

Reggio Calabria, 1 January 2008

Signature
Francesco Buccafurri

**Figure 7.** The result of opening the file `delegation.bmp`

For the signature verification, the official software[4] distributed by the CNIPA [10] has been exploited. The same result is obtained by any signature verification software (like Aloaha Sign [4]).



Verifica
Nome file Filename
C:\Documents and Settings\user\Desktop\delegation.htm.p7m

Esito verifica Verification result
Firma corretta The signature is OK

Certificati firmatari
Francesco Antonio Buccafurri Signer's name
Certificato credibile (CA accreditate) The Certification Authority is accredited
Certificato valido fino a giovedì 3 marzo 2011 01:00:00 Cert. expiration date: 3 March 2011
Certificato non revocato; informazioni di revoca valide da domenica 28 settembre 2008 08:02:42 fino a domenica 28 s
The certificate is not revoked
Certificato per il non ripudio
Certificate description: non repudiation

**Figure 8.** The result of the verification process

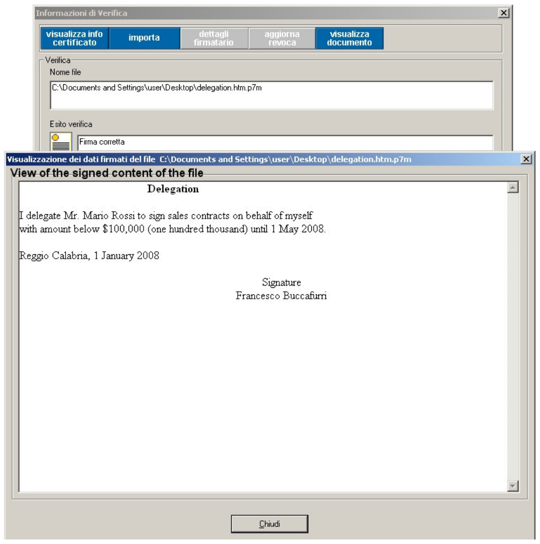
Moreover, by clicking on "Display document", the signed document is shown (Figure 9). Surprisingly, the document appears dramatically different from the signed one. It contains the text *T*, giving the assistant the delegation to sign sales contracts with amount below $100,000, instead of the $1,000 approved by Prof. Buccafurri.

In the next two sections, we prove that the attack succeeds also if we replace the bitmap format by other widely used image formats, like TIFF and PDF.

[4] Since the language of the software is Italian, we have translated into English the most relevant information supplied by the software.

## 3.2 Attack on TIFF

Now, we describe how the attack is performed in case the format *A* is TIFF, an image file format widely supported by scanning, faxing, image-manipulation, word processing and optical character recognition (OCR) applications. TIFF files support many features such as compression, multi-page graphics and so on.



Verifica
Nome file
C:\Documents and Settings\user\Desktop\delegation.htm.p7m

Esito verifica
Firma corretta

Visualizzazione dei dati firmati del file C:\Documents and Settings\user\Desktop\delegation.htm.p7m
View of the signed content of the file

**Delegation**

I delegate Mr. Mario Rossi to sign sales contracts on behalf of myself
with amount below $100,000 (one hundred thousand) until 1 May 2008.

Reggio Calabria, 1 January 2008

Signature
Francesco Buccafurri

**Figure 9.** The document shown by the verification software

In order to support such features, the TIFF format relies on a very flexible file structure, in such a way that the information about the image is referenced by an Image File Directory (IFD), that is an array of fields describing the features of the image, i.e. resolution, number of colors used, compression, image data, etc. Each feature is then suitably encoded in a separate IFD entry. Moreover, IFDs are arranged in a linked list, hence in case the file includes multiple images (such as pages of facsimile transmission, or scanned book pages), each is represented by a subfile described by a different IFD.

In detail, a TIFF file [3] begins with a 4-byte header, (either `49492A00` or `4D4D002A`, in hexadecimal) specifying the byte order (either little-endian or big-endian, respectively) used to encode numeric values. The header is followed by two bytes representing a pointer to the first IFD (For the sake of simplicity, assume that the file has just one IFD).

The attacker proceeds as follows: he inserts the opening HTML comment (`<!--`) just after these two bytes and then modifies them by increasing the encoded value by 4, in order to take into account the insertion of the 4-byte string `<!--`. Clearly, he has to be aware of the byte order (encoded in the header) in order to tamper the file correctly.

Then, the attacker finds the IFD entries containing the offsets (with respect to the beginning of the file) of the locations where the bytes representing the actual data of the image (i.e. the pixelmap and the colormap) are stored. Hence, he increases each of these offsets by 4 in order to make them point to the correct locations. This operation has to be iterated for the entries encoding the following tags

*StripOffsets*, *XResolution*, *YResolution*, *ColorMap* and, in case the image pixels are encoded in more than one location within the file, for all the IDF entries encoding any image data offset.

Finally, the attacker concatenates the HTML code containing the malicious content at the end of the tampered file.

### 3.3 Attack on PDF

We have implemented the attack also in case the format *A* is PDF, a file format created by Adobe Systems in 1993 for document exchange and used for representing two-dimensional documents in a way that is independent of the application software, hardware, and operating system.

According to [2], the structure of a PDF file (here we consider the simple and common case of a PDF file with no incremental update) is the following:

- **Header.** The file begins with an ASCII one-line header identifying the version of the PDF specification to which the file conforms.
- **Body.** The body contains the document objects included in the file.
- **Xref.** A cross-reference table containing pointers and other information about the objects included in the body of the file.
- **Trailer.** A closing section, ending with the sequence `%%EOF`, giving the location of both the cross-reference table and special objects within the body of the file.

The attacker behaves as follows: He starts by inspecting the header. It is of the form `%PDF-X.Y`, that is a Postscript language comment, i.e. an ASCII sequence starting with the symbol `%` and ending with an End-of-Line (EOL) ASCII code, where the sequence `X.Y` specifies the version of the PDF format. The current version is 1.7 (corresponding to Adobe Acrobat 8). Observe that our attack is not dependent on the version of the PDF file, i.e. it can be used also on older versions of PDF.

The attacker embeds the opening HTML comment in a Postscript comment `%<!--` (ending with a non-printable EOL character) and inserts it just after the header. In case a PDF file contains binary data, the header should be followed by a Postscript comment line containing at least four bytes, each having a value 128 or greater. In such a case, the comment `%<!--` (including the final EOL code) should follow it. This ensures proper behavior of file transfer applications that inspect data near the beginning of a PDF file in order to determine whether to treat the file's contents as text or as binary.

Then, the attacker finds the location of the cross-reference table by inspecting the tail of the file. There, the Trailer includes two lines just before the end of the file, marked by the sequence `%%EOF`. The former is the keyword `startxref`, the latter is the offset (encoded as an ASCII string) of the cross-reference table. The attacker increases the offset value by 6 (in order to take into account the previous insertion of the opening HTML comment) and then changes all the *in-use* entries of the cross-reference table by adding 6 to the value encoded by the offset field. This ensures that the PDF viewer will correctly decode the file. Finally, the HTML code containing the malicious content is concatenated to the tampered file.

## 4. A Possible Solution

In this section a discussion on a possible solution to the attack scheme described above is given.

Basically, we propose to include the MIME Content-type of the document to be signed into the cryptographic message in such a way that the integrity of both the document (the file) and the file format (associating the file with the intended viewer) is guaranteed.

First, we need to give some detail about the PKCS#7 format. It supports several different content types: *data*, *signed data*, *enveloped data*, *signed-and-enveloped data*, *digested data*, and *encrypted data*.

The data content type represents a sequence of bytes. The encrypted-data content type consists of encrypted content of any type. The digested-data content type consists of content of any type and a message digest of the content. The signed-data content type consists of content of any type and encrypted message digests of the content for zero or more signers and it is used to represent digital signatures. The enveloped-data content type is intended to represent *digital envelopes*, combining encrypted data sent to one or more recipients and the information (the *content-encryption keys*) needed by each recipient in order to decrypt the content. Finally, the signed-and-enveloped-data content type represents digital envelopes providing data with "double encryption", i.e. an encryption with a signer's private key followed by an encryption with the content-encryption key.

Any of the content types defined in *PKCS#7* can be enveloped for any number of recipients and signed by any number of signers in parallel. The signed-data content type is intended to be used for digital signatures, and it constitutes the basis upon the cryptographic message is built. Such a content type consists of (i) a given content of any of the types defined in *PKCS#7* and, for each signer, (ii) both an encrypted message digest of the content (i.e. of the document) representing the signer's digital signature on the content, and (iii) other signer-specific information (concerning, for example, certificates and certificate-revocation lists). Additional information can be signed in order to authenticate attributes other than the content, such as the signing time.

In detail, the signed-data content type consists of the following information:

a. A list of the message-digest algorithms used by the signers (this information is optional and it is used to make one-pass signature verification easy).
b. The content that is signed. It can have any of the defined content types.
c. An optional set of X.509 certificates and PKCS#6 extended certificates.
d. An optional set of certificate-revocation lists used to determine whether or not the certificates referenced by the above item are "hot listed"

(because, for instance, they have been either revoked or suspended for some reason and thus they are not trustable anymore).

e. A set of per-signer information:
   (1) The certificate issuer's distinguished name and serial number;
   (2) An identifier specifying the message-digest algorithm (e.g. SHA-1) used by the signer under which the content and authenticated attributes (if any, see the next item) are digested;
   (3) An optional set of PKCS#9-compliant attributes that are signed by the signer (e.g. the signing time);
   (4) An identifier specifying the encryption algorithm under which the message digest and the associated information are encrypted with the signer's private key;
   (5) The result of encrypting the message digest and the associated information with the signer's private key (this information is the signer's digital signature);
   (6) An optional set of PKCS#9-compliant attributes that are not signed such as countersignatures, i.e. signatures to be associated with another signature.

The simple solution here proposed is to choose a suitable MIME Content-type value [20], i.e. corresponding to the intended format of the document to be signed. For instance, if it is a TIFF image (`.tif`) the correct value is *image/tiff*[5].

Such a value must be included into the PKCS#7 cryptographic message by suitably encoding it into the authenticated attributes (item e.3 above).

In detail, according to [35], the chosen MIME Content-type value corresponding to the format of the file to be signed should be encoded into the PKCS#9-compliant attribute type *allegedContentType*. Next, both the document digest and such authenticated attributes are encrypted with the signer's private key. Finally, a suitable digital-signature verification software should be aware of such an additional information in order to check the integrity of both the document and the file format. Hence, if an attacker renames the cryptographic message file, the verification software, by extracting the signed Content-type value, will correctly display the document, thus disarming the attack.

In case the cryptographic message is formed according to Cryptographic Message Syntax (CMS) [24], then our proposal is that the information about the document format (represented by the MIME Content-type value) should be included into the *content-hints* attributes [22]. Such attributes are only intended for encoding optional information (such as the MIME type) defining the document format [15].

In detail, according to [15], the *contentType* field should be set as "id-data" and the *contentDescription* should contain the MIME *Content-Type* header value specifying the intended presentation format.

[5] The list of Content-types is maintained by the Internet Assigned Numbers Authority (IANA), at *http://www.iana.org/assignments/media-types/*.

Observe that, though technical specifications and security recommendations [15], [17], [18] address the theoretical problem of misunderstanding (during the verification task) the format of the document which has been digitally signed, they consider no concrete attack scheme which is based on the filename-dependent vulnerability presented in the paper. Moreover, they regard *content-hints* attributes as optional ones. Likewise, PKCS#7 syntax considers authenticated attributes as optional parameters.

However, a technically simple solution does not correspond in general to a practically simple solution. Indeed, in this case, what about technical rules about the usage of PKCS#7 in the signature process? What about signature generation and verification software currently provided by Certification Authorities?

Maybe a less simple, more heuristic, solution could be more feasible, like a parser-based approach allowing us to detect patterns identifying the attack.

## 5. Conclusions

The importance of encryption-based digital signature is nowadays universally known, due to the revolution that such a mechanism has induced on the role of electronic documents in both public and private organizations. In fact, digital signature represents at the moment the only valid method to give signed electronic documents probative value at least as traditional documents with handwritten signature. The above claim has a full counterpart with the current law system of most countries, so that the process of document dematerialization has been already started relying on the current infrastructures as well as the current juridical regulations, with strong attention towards common interoperability rules. On the basis of the above observations we may easily realize that the issue regarding the vulnerabilities of digital signature is particularly important. This paper presents a new attack on digital signature succeeding when the signature is generated just from the bits contained into the document. Indeed, the integrity of their intended meaning, corresponding to what the user have signed, is not guaranteed. The conclusion is thus that the encryption transformation, in order not to suffer from the presented vulnerability, should process more than the document content. The paper presents a basic yet effective solution, based on mandatorily signing also the MIME Content-type. Stronger solutions, based on the parsing of the document being signed could be of course considered.

## References

[1] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. "Authentication and delegation with smart-cards". In *TACS'91: Selected papers of the conference on Theoretical aspects of computer software*, pages 93–113. Elsevier, 1993.

[2] Adobe Systems Inc. "PDF reference, fifth edition: Adobe portable document format version 1.6", 2004. http://www.adobe.com/devnet/pdf/pdf_reference.html.

[3] Adobe Systems Inc. "TIFF 6.0 specification", 1992. http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf.

[4] Aloaha Sign. http://www.aloaha.com/.

[5] I. Z. Berta, L. Buttyan, and I. Vajda. "Mitigating the untrusted terminal problem using conditional signatures". In *ITCC '04: Proc. of the Int. Conf. on Inf. Technology: Coding and Computing*, Vol. 2, page 12. IEEE Computer Society, 2004.

[6] BMP file format entry from Wikipedia. http://en.wikipedia.org/wiki/BMP_file_format/.

[7] F. Buccafurri and G. Lax. "Hardening digital signatures against untrusted signature software". In *Proceedings of the 2nd IEEE International Conference on Digital Information Management (ICDIM'07)*, pages 159–164, 2007, IEEE Press.

[8] F. Buccafurri and G. Lax. "Signing digital documents in hostile environments". *Int. J. Internet Technology and Secured Transactions*, (to appear).

[9] D. Clarke, B. Gassend, T. Kotwal, M. Burnside, M. van Dijk, S. Devadas, and R. Rivest. "The untrusted computer problem and camera-based authentication". In *Proc. of the 1st Int. Conf. on Pervasive Computing*, volume 2414 of LNCS, pp. 114–124. Springer, 2002.

[10] CNIPA. http://www.cnipa.gov.it.

[11] CNIPA. "Italian technical rules (DPCM 13/01/2004)". http://www.cnipa.gov.it/site/_files/DPCM%20040113_v2.pdf.

[12] CNIPA. Personal communication, April 2008.

[13] R. Cramer and V. Shoup. "Signature schemes based on the strong RSA assumption". In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*, pages 46–51, New York, NY, USA, 1999. ACM Press.

[14] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: "A strengthened version of RIPEMD". In *Fast Software Encryption*, pages 71–82, 1996.

[15] ETSI. "Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES)". Technical Specification TS 101 733 V1.7.4., 2008.

[16] European Commission. "ITSEC - Information technology security evaluation criteria: Preliminary harmonised criteria". Document COM(90) 314, Version 1.2, 1991.

[17] European Committee for Standardization. "Security requirements for signature creation applications". CWA14170, 2004.

[18] European Committee for Standardization. "General guidelines for electronic signature verification". CWA14171, 2004.

[19] FESA. http://www.fesa.eu.

[20] N. Freed, N. Borenstein. "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies". IETF RFC 2045, 1996.

[21] H. Gobioff, S. Smith, D. Tygar, and B. Yee. "Smart cards in hostile environments". In *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, pages 23–28, 1996.

[22] P. Hoffman. "Enhanced Security Services for S/MIME". IETF RFC 2634, 1999.

[23] R. Housely, W. Ford, W. Polk, D. Solo. "Internet X.509 Public Key Infrastructure". IETF RFC 2459, 1999.

[24] R. Housley. "Cryptographic Message Syntax". IETF RFC 3852, Vigil Security, 2004.

[25] A. Jøsang, D. Povey and A. Ho, "What You See is Not Always What You Sign". In *Proceedings of the Australian Unix User Group Symposium (AUUG2002)*, Melbourne, 4-6 September, 2002.

[26] K. Kain, S.W. Smith and R. Asokan. "Digital signatures and electronic documents: a cautionary tale". In *Proceedings of The Sixth Joint Working Conference on Communications and Multimedia Security*, September 26-27, Portoroz, Slovenia, pp. 293–308, Kluwer, 2002.

[27] B. Kaliski. "PKCS#7: Cryptographic Message Syntax". IETF RFC 2315, RSA Laboratories, 1998.

[28] B. Lee and K. Kim. "Fair exchange of digital signatures using conditional signature". In *Symposium on Cryptography and Information Security*, 2002.

[29] T. Matsumoto. "Human-computer cryptography: An attempt". In *ACM Conference on Computer and Communications Security*, pages 68–75, 1996.

[30] NIST/NSA. "Fips 180-2 secure hash standard (SHS)", 2002.

[31] PDF/A Competence Center. http://www.pdfa.org.

[32] R. L. Rivest. "Issues in cryptography". In *Computers, Freedom, and Privacy Conference*, 2001.

[33] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". *Commun. ACM*, 21(2):120–126, 1978.

[34] P. Rybar. "Specifying the content and formal specifications of document formats for QES". National Security Authority, Department of Information Security and Electronic Signature, 2007. http://www.nbusr.sk.

[35] RSA Laboratories. "PKCS #9 v2.0 Amendment 1", 2003.

[36] B. Schneier and A. Shostack. "Breaking up is hard to do: Modelling security threats for smartcards". In *Proc. USENIX Workshop Smart Card Technology*, 1999.

[37] T. Stabell-Kulo, R. Arild, and P. H. Myrvang. "Providing authentication to messages signed with a smart card in hostile environments". In *Proceedings of the USENIX Workshop on Smartcard Technology*, 1999.

[38] P. Wayner. *Disappearing Cryptography, - Information Hiding: Steganography and Watermarking. Software Engineering and Programming*. Morgan Kaufmann Publishers, 2002.

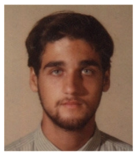
[39] B. Yee and J. D. Tygar. "Secure coprocessors in electronic commerce applications". In *Proceedings of the first USENIX Workshop on Electronic Commerce*, pages 155–170, 1995.

## Author Biographies

**Francesco Buccafurri** is a full professor of computer science at the University "Mediterranea" of Reggio Calabria, Italy. In 1995 he took the PhD degree in computer science at the University of Calabria. His research interests include deductive-databases, knowledge-representation and non-monotonic reasoning, model checking, information security, data compression, data streams, agents,

P2P systems. He has published several papers in top-level international journals and conference proceedings. He serves as a referee for international journals and he is member of a number of conference PCs.

**Gianluca Caminiti** holds a PhD degree, received in March 2006 from the University "Mediterranea" of Reggio Calabria, Italy. In 2002 he took the Laurea degree in Electronics Engineering. His research interests cover the field of artificial intelligence, including multi-agent systems, logic programming, knowledge representation and non-monotonic reasoning. He has published a number of papers in top-level international conference proceedings. He has been involved in several national and international research projects.



**Gianluca Lax** is an Assistant Professor of computer science at the University "Mediterranea" of Reggio Calabria. In 2005 he took the PhD degree in computer science at University of Calabria. His research interests include data reduction, data streams, user modelling, P2P systems, e-commerce and information security. He is also author of a number of papers published in top-level international journals and conference proceedings.