Applied Computing and Informatics 14 (2018) 145–158

Contents lists available at ScienceDirect

Applied Computing and Informatics

journal homepage: www.sciencedirect.com

A method for verifying integrity & authenticating digital media

Martin Harran [a], William Farrelly [a], Kevin Curran [b,*]

[a] Letterkenny Institute of Technology, Donegal, Ireland
[b] Ulster University, Derry, United Kingdom

ARTICLE INFO

Article history:
Received 6 March 2017
Revised 22 May 2017
Accepted 24 May 2017
Available online 31 May 2017

Keywords:
Secure authentication
Integrity
Digital certificates
Security
Network security

ABSTRACT

Due to their massive popularity, image files, especially JPEG, offer high potential as carriers of other information. Much of the work to date on this has focused on stenographic ways of hiding information using least significant bit techniques but we believe that the findings in this project have exposed other ways of doing this. We demonstrate that a digital certificate relating to an image file can be inserted inside that image file along with accompanying metadata containing references to the issuing company. Notwithstanding variations between devices and across operating systems and applications, a JPEG file holds its structure very well. Where changes do take place, this is generally in the metadata area and does not affect the encoded image data which is the heart of the file and the part that needs to be verifiable. References to the issuing company can be inserted into the metadata for the file. There is an advantage of having the digital certificate as an integral part of the file to which it applies and consequently travelling with the file. We ultimately prove that the metadata within a file offers the potential to include data that can be used to prove integrity, authenticity and provenance of the digital content within the file.

1. Introduction

More of today's communications are moving to exclusively digital format with no hard copy backups being kept. This has implications regarding integrity, authentication and provenance in various areas such as litigation where it is necessary that both sides are satisfied with the integrity of digital evidence or in the insurance industry where claims can succeed or fail on very detailed terms and conditions; it may be necessary to know exactly what the terms and conditions were at the time the policy was created [1]. There is also a need to verify the terms of a contract that applied when the contract was agreed and establish dates of original creation when copyright issues arise regarding digital content [2]. Likewise, real world requirements exist to establish prior knowledge before signing Non-Disclosure Agreements. An important aspect of this is that the raw data content of a file can be verified even when the metadata is altered due to the file moving across operating systems or devices.

Widespread and easily available tools have become common for video synthesis. It is important to ensure the authenticity of video uses such as in courts of law, surveillance systems, advertisements and the movie industry. There is therefore a large body of research in the ares of video authentication and tampering detection techniques [3–5]. There is also research using Darwin Information Typing Architecture in protecting the integrity of digital publishing applications [6].

Although there are many tools available to create, encrypt and extract data, there is relatively little available in the area of establishing integrity, authentication and provenance. For instance, [7] creates and issues a certificate containing a SHA 256 hash value of the submitted media file along with the user details and a timestamp. That certificate is in turn verified by a digital certificate issued by leading certification authority Comodo. The problem here however is that the original file and the certification are separate entities and could easily become separated during distribution or circulation of the file. In general, wrapping the file and its certification inside an outer envelope is no real guarantee that the field and its certification will remain together. Inserting the certification into the file is the only apparent reliable method and is the approach used by leading companies such as Microsoft and Adobe for digital signing of Office and PDF documents [8]. A limitation with inserting the certificate inside the file is that it

* Corresponding author.
E-mail address: kj.curran@ulster.ac.uk (K. Curran).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.aci.2017.05.006
2210-8327/

must follow the metadata specification set out for the particular file type which means that distinct software applications or components have to be developed for every file type.

This research outlines a method in which integrity, authentication and provenance can be established for raw data within a file even when the metadata attached to it has changed. We also achieve this in a manner in which a digital proof stamp can be made visual to help achieve virality for the end users.

2. Digital fingerprinting

Digital fingerprinting is based on the use of a mathematical function to produce a numerical value where the function takes an arbitrary length of data as its input and outputs a numerical value of a specified length; 128 bits (16 bytes), 256 bits (32 bytes) and 512 bits (64 bytes) are typical lengths. The output value of such a function is generally referred to as a *hash value* or simply *hash*. In order to be useful, the function used to produce a digital fingerprint must be quick to use but in order to be trustworthy for authentication purposes, it must also meet certain security requirements [9]. For this reason, digital fingerprinting uses *cryptographic hash functions*; the authors describe the general security properties of these functions as preimage resistance, second preimage resistance (or weak collision resistance) and collision resistance (or strong collision resistance) where:

- Preimage resistance means that for a hash function H and an output value of z, it is *computationally infeasible*[1] to find an input value m such that $z = H(m)$. This property is also known as *one-wayness*.
- Second preimage resistance means that given an input m_1, it is computationally infeasible to find a different message m_2 where $H(m_1) = H(m_2)$. This is also known as *weak collision resistance*.
- Collision resistance means that it is computationally infeasible to find two different inputs where m_1 *and* m_2 where $H(m_1) = H(m_2)$. This is also known as *strong collision resistance*.

Ref. [10] stated that collision resistance implies second preimage resistance but does not guarantee preimage resistance. Ref. [11] pointed out that that the validity of those claims is limited to specific definitions of the various terms. The subtleties involved, however, have no impact on this project as only second preimage resistance applies in regard to the use of digital fingerprints for file authentication - a potential attacker has access to both the original input (the file) and the output (the hash value for the file).

A wide range of algorithms have been developed to meet the requisite security requirements as well as having the underlying performance requirement of computational speed. Two of the most commonly encountered functions are MD5 and the SHA family of functions. MD5 (Message Digest algorithm 5) was developed by Ref. [12] who had previously developed earlier versions MD2 – MD4. It became extremely popular following its release but beginning in 1996, vulnerabilities were increasingly identified in regard to collision resistance. Ref. [13] stated that although practical applications were not yet threatened, it "comes rather close" and advised users to move to other algorithms. Eight years later, Ref. [14] announced collisions based on the full hash. Four years after that, Ref. [15] based MD5 collision vulnerabilities to create a rogue digital certificate which would be accepted by all common browsers. The credibility of MD5 was effectively terminated in December that year by the release of a Vulnerability Notice by US-CERT [16] which stated that "it [MD5] should be considered cryptographically broken and unsuitable for further use". Although MD5 is still adequate for checking file consistency and other non-secure applications, the known weaknesses should have effectively ended its use for digital signatures. It is still in widespread use, however, as late as 2012 when it was used to fake a Microsoft digital signature in the *FLAME* malware attack [17]. Rivest has produced an MD6 version but after submitting it to the NIST open competition for SHA-3, he withdrew it stating that it was not ready for use [18]. Following the decline of MD5, the predominant algorithms in use today are members of the SHA family.

2.1. Secure hash algorithm

Secure Hash Algorithm (SHA) is a family of cryptographic hash functions developed by the US National Security agency (NSA) and published as standards by the US National Institute of Science and Technology (NIST). It is the required algorithm for secure applications used by US government agencies. An important feature of the SHA algorithms is that they implement an *avalanche effect* which means that a minor change to the input cascades into a major change in the output value.

The first version, SHA-1 produces a 160-bit value and was released in 1993; it was, however, withdrawn shortly after publication due to an undisclosed vulnerability and a modified version was released two years later [19]. In 2002, NIST published the SHA-2 family of functions. Unlike SHA-1 with hash size fixed at 160 bits, SHA-2 is offered in six versions producing a range of output sizes from 224 to 512 bits of which the most widely used are SHA-256 and SHA-512. Like MD5 and SHA-1, the SHA-2 functions are based on the Merkle–Damgård construction. The algorithm used for SHA-256 is:

1. A 256-bit buffer is created, made up of 8 × 32-bit words which are initialised with the first 32 bits of the fractional parts of the square roots of the first 8 primes.
2. A 64 element table of constants is prepared using the first 32 bits of the fractional parts of the cube roots of the first 64 primes

The input is padded with an initial bit '1' and the length of the original input expressed as a 64-bit integer, separated by the required number of zeros required to make the message length, including the padding, a multiple of 512 bits.

3. Each 512-bit block is processed through 64 rounds where each round involves a series of operations comprised of bitwise operations and modular addition.
4. The value of the buffer on completion of each block is the initial value for the following block; at the end of the final block, the buffer contains the hash value.

Despite offering better security and faster performance, SHA-2 was only adopted slowly over the next three years with SHA-1 remaining in extensive use. The main contributory reasons were that SHA-2 was not supported on systems using Windows XP (SP2) or older, there was no perceived urgency as no collisions had yet been found in SHA-1 and SHA-256 is about 2.2 times slower than SHA-1 though some of that drop can be reduced by a move to SHA-512 on 64-bit hardware and operating systems [20].

Ref. [21] revealed collision attacks on the full SHA-1 function. This work did not completely undermine the practical reliability of the function – it was a *strong collision attack* which did not impact on the preimage – but the following year, NIST instructed

[1] An expression used in cryptology to signify that, given enough time and resources, it may be theoretically possible to decrypt the result of a cryptographic function but the time and resources actually available make it impractical in any meaningful way.

federal agencies to stop using SHA-1 as soon as practical and stated that after 2010, SHA-2 would be compulsory for digital signatures, digital time stamping and other applications requiring collision resistance [22]. Although SHA-1 is still in extensive use and is still secure for most practical purposes, applications that continue using it are likely to run into ever increasing problems as software developers drop support from new versions of their applications; Microsoft, Chrome and Mozilla, for example, have all announced that SHA-1 will not be supported in browser versions beyond 2016.

No significant attack on SHA-2 has ever been demonstrated. Nevertheless, in what can be regarded as a pre-emptive move, NIST launched an open competition in 2007 for the development of SHA-3. The reason for adopting an open competition approach was to find an alternative to the Merkle–Damgård construction as that had been the basis of the now discredited MD5 and SHA-1 as well as SHA-2. The eventual winner, *KECCAK* using what is known as the *sponge* construction, was announced in 2012 and the new standard published three years later [23]. The pre-emptive nature of SHA-3 is clearly indicated by NIST whose current policy on hash functions [24] states that there is no need to transition applications from SHA-2 to SHA-3.

2.2. RSA

The original concept of a digital authentication scheme was first proposed by Diffie and Hellman [25], who revolutionised cryptography by proposing a method of creating a shared secret over public or unsecured channel which would enable traditional *symmetric* cryptography to be used electronically. In their paper, they also discussed the idea of a system of *public key cryptography* based on *public* and *private* keys and also the need for *one-way authentication* but did not propose solutions for either of these. Two years later, such a solution was developed at Massachusetts Institute of Technology (MIT) by three researchers [12] which became known by the initial letters of their surnames, RSA (Rivest- Shamir- Adleman). The RSA public/private key algorithm is based on the difficulty of factoring large numbers and produces a private key and a public key where a message encrypted using the public key can be decrypted using the private key but calculating the private key from the public key is computationally infeasible. This is known as *asymmetric* encryption. The RSA algorithm is too slow for encrypting lengthy messages but it can be used to create a shared secret which can then be used for high-speed symmetrical encryption.

2.3. Digital signatures

Ref. [12] also proposed that authentication of content, whether encrypted or in plain-text, could be achieved by using the keys to encrypt and decrypt a digital fingerprint. The overall process, a message accompanied by a hash of the message encrypted with the sender's private key constitute what is known as a *digital signature.* This process fulfills the twin requirement of authentication i.e. identification and non-repudiation. If the calculated hash value matches the decrypted hash value from Alice, then Bob can be confident that the message was indeed sent by her and the message has not changed in transit. Alice, in turn, cannot deny sending the message provided that ker private key was genuinely used to encrypt the hash. That was the main outstanding problem – establishing a trustworthy way to distribute public keys. The processes are open to *man-in-the-middle* attacks where a third party could substitute their own public key for that belonging to Alice and send a different file and hash to Bob and he will be duped into believing it is a genuine message from Alice.

Reliable distribution of public keys can be achieved through the use of *digital certificates*, also known as *public key certificates,* which are electronic documents that verify the owner of a public key. These certificates are issued by a *Certification Authority* (CA), an organisation which specialises in the business of issuing certificates, usually on a commercial basis and which can be trusted by recipients. The certificate is a binding between an entity and its associated public key and contains information about the issuing CA as well as information about the person or organisation to whom the certificate applies. This information is generally presented in the format of an X.509 certificate. Digital certificates issued by third-party certification authorities are the backbone of secure Internet connections based on Transport Layer Security (TLS), formerly known as Secure Sockets Layer (SSL); a secure connection is most recognisable when a website uses Hypertext Transfer Protocol Secure (HTTPS) and a padlock icon appears in the URL or at the bottom of the browser. The digital certificate for the site can be viewed by clicking on the padlock. Until comparatively recently, secure connections using SSL/TLS were primarily used for webpages involving financial transactions but they are coming into wider use in popular sites such as Google and social media sites such as Facebook and Twitter.

The highest level of CA is known as a *Root CA* and public keys for these root CAs have to be authenticated using the issuer's own public key. These keys therefore have to be readily available and are built into operating systems and browsers. In addition to root trusted CAs, a range of other categories is included such as Trusted Publishers, Trusted People and Other People. These are managed through a hierarchical structure where higher levels of CA can delegate authority to lower level CAs for specific uses. This requires a *chain of trust* between the root CA and the ones descended from it.

Despite this chain of trust and a proliferation of intermediary CAs and digital certificate resellers, the market in reality is dominated by a small number of organisations. In the TLS/SSL market for April 2016 for example, just 5 players issued 89% of the certificates – Comodo had 41% of the market, Symantec (including Verisign) had 26%, Godaddy had 12% and GlobalSign had just under 10%. No other player had more than 1%. Digital certificates can be used for a wide range of applications ranging from email to financial transactions and CAs offer certificates for specific uses. Various types of use require different levels of security in regard to owner identification and CAs typically offer different *classes* of certificate to reflect this, ranging from those where minimal checking is done on the applicant to those where the applicant must undergo a rigorous identification process. Unfortunately, there is no standardised procedure for these classes and their implementation varies among issuers. Extended Validation (EV) is an enhanced procedure for validating the owners of websites, developed through a voluntary group of certification authorities [25]. The need for that improved level of authentication was demonstrated in 2011 when an intruder hacked into the systems of an Italian reseller of Comodo digital certificate and directly into the systems of leading Dutch certification authority DigiNotar and managed to get digital certificates that appeared to be issued in the names of various high-profile[2] organisations [26]. Comodo identified the problem with the Italian reseller within hours and were able to revoke the certificates and notify customers thus minimising damage to their own reputation. In the case of DigiNotar, however, the company's systems were found to be so insecure that the root certificate had to be marked as untrusted which in turn invalidated all certificates issued under the root and, despite the intervention of the Dutch government, the company was forced into bankruptcy. These attacks, however, were an exemplification of the adage that any system is only as secure as the procedures it implements – they did not

[2] Reputed to include Gmail, Hotmail and Yahoo Mail but that has not been confirmed.

undermine the underlying principle of either RSA or digital certificates. RSA –based digital certificates remain the main technique for digital authentication in use today.

3. Utilising metadata for authenticity

A core research question we had was whether "Is it possible to add additional data to the existing metadata of a file in such a way that the added data that can be used to check the integrity, authenticity and provenance of the digital content within the file?"

This necessitated reviewing existing methods of editing and inserting metadata into files to determine what limitations exist and whether existing applications could be adapted to fulfil the requirement rather than developing a new software artefact from scratch. There is limited native support in operating systems for metadata editing. Most Linux distros do not have any native means of viewing metadata although applications such as ExifTool can be installed. The most notable exception is Ubuntu desktop which displays some of the metadata similar to Mac OS X, again uneditable. An example can be seen in Fig. 1.

Unlike Mac OS X and Linux, Windows Explorer does provide the facility to edit several tags which have been marked with an asterisk in Fig. 1. Editing in Windows Explorer, however, causes radical changes to the structure of the JPEG. There is a wide range of imaging processing and management applications that handle editing and inserting metadata to various degrees. The most powerful and flexible application available is the open source ExifTool by Phil Harvey (available from www.sno.phy.queensu.ca/~phil/exiftool/). It covers a wide range of file types, executables as well as image files and offers extensive functionality for both viewing and editing metadata. It is available both as a Perl library and as a command line application and it is the application of choice for Linux [27] due to the fact that the author has published the code and invited the open source community to participate in its ongoing development.

A number of Windows front-ends for ExifTool have been developed such as XnViewMP (available from http://www.xnview.com/en/xnviewmp/) but these have limited editing facilities. ExifTool is at its most powerful at the command line where it can be used to view and/or edit not only all tags for all known image formats but also for a wide variety of other file types including executables. Open source Exiv2 by Andreas Huggel (available from http://www.exiv2.org/getting-started.html) is similar to Exif tool but is published as a C++ library and command line utility. It offers very similar functionality to ExifTool but does not cover as wide a range of file types.

EXIF is supported by many Graphical User Interface (GUI) applications for reading metadata but editing facilities are generally limited, especially where tags need to be inserted. This author did find a couple of shareware applications for doing so – Exif Pilot (available from www.colorpilot.com/exif.html) and Photome (available from www.photome.de/) but their functionality is restricted.

IPTC/XMP metadata is generally well supported. Due to the relationship between IPTC and Adobe, the latter's flagship file management application, *Bridge*, gives extensive support to IPTC across the full range of Adobe applications - Acrobat, ImageReady, InDesign and PhotoShop as well as JPEG and PNG – and is the application of choice for managing IPTC and XMP. Other applications support IPTC to various degrees; several freeware applications such as IrfanView and XnViewMP offer good view-only functionality but Gimp, the main open source alternative to Photoshop, offers some limited IPTC/XMP editing facilities.

There are also some API libraries available such as *Windows Imaging Component* in Windows .Net Framework which support

Fig. 1. Metadata edit and view - Windows Explorer.

editing and inserting metadata in both JPEG and PNG images. Problems have being found with that library [28]. Apple OS X also has a programmatic way of accessing metadata using Spotlight and the tag *kMDItem* although detailed study of that was not carried out.

In regard to the proposed software artefact, careful consideration was given to using either ExifTool or Exiv2 which are available as opensource libraries but a unanimous decision was made to write a new application form scratch. The main reason for this was that both applications/libraries were far more wide ranging than needed for the proposed application and it was difficult to just extract the specific functionality needed; also, there was the added complexity with ExifTool of having to access a Perl module from within C++. The most decisive factor, however, was that neither of these libraries would support inserting a digital certificate. It would be easier to add in new code for handling metadata rather than trying to include 3rd party modules. Both Microsoft Office and Adobe Acrobat provide the facility to insert digital certificates into their relative document types and the new 3MF file type for 3D printing has the facility built in. These are all for proprietary file types, however, based on XML structures and are not applicable to JPEG, PNG or plain text files. Despite extensive searching, no off-the-shelf library or application could be found to insert data into its own section in these file types. Developers and programmers have sought to add data other than metadata into JPEG files in particular, sometimes for good reasons such as watermarking images and other times for nefarious ones; although an image file cannot execute any code, the Zeus banking Trojan used a JPEG file to disguise and distribute a crucial configuration code. These approaches, however, involve changing the image data to add watermarking or by stenographic techniques using the *Least Significant Bit*. Also, stenographic techniques are used to *hide* information whereas we would like the inclusion of a digital certificate to be as visible as possible. There is no native support for editing metadata in PNG files in any of the main operating systems (Windows, OS X, Linux). A number of applications were tried including

PNGCrush and TweakPNG but the only one that was found to be really worthwhile was ExifTool.

We conclude that a digital certificate needs to be inserted into a unique field due to the risk of it being overwritten by other metadata editing applications. In effect, that meant that existing applications and libraries were of little use and an application that could handle inserting a digital certificate would have to be developed from scratch. That required a more in-depth analysis of the two image types selected. Details of that work follow in the next section.

3.1. JPEG

The first two bytes in a JPEG file are the file signature with a value of 0xFFD8. This is known as the Start Of Image (SOI) marker and it is matched at the end of the file by a two-byte End Of Image (EOI) marker with a value of 0xFFD9. Inside these markers, the file contains the encoded image data preceded by a number of segments as shown in Fig. 2. All segments are in the form where the first two bytes have a tag identifying the type of segment and the next two bytes indicate the length of the segment, excluding the bytes for the tag but including these two. In the example shown in Fig. 3, the first two bytes – 0xFFD8 - are the JPEG File Signature. Bytes 3 and 4 are the segment tag - 0xFFE1 - and the length of the segment is 0xEDF6 bytes. The rest of the segment contains the segment data. As the length has to be expressed in two bytes, the maximum length of a segment excluding its tag is 64K. Some segments can, however, occur multiple times which allows that limit to be overcome. Although there are a range of defined segment tags, there is nothing to prevent the use of a custom segment. Provided the format is followed, a well behaved application which does not recognise the tag will simply use the stated length to jump to the next segment. The first group of segments after the SOI markers hold the metadata for the file which will be discussed in detail in the next section as this is the primary focus of the software solution devised.

The metadata segments are followed by segments holding data relevant to the compression algorithm. JPEG is a lossy compression technique which uses quantization and Huffman coding. Depending upon the way the image was created or edited, other segments may be present for parameters such as Restart Interval (DRI). Compression information segments are of no direct interest in this project but the fact that they can be restructured has to be taken into account; for example, inserting an extra metadata field in various applications may split a single DQT or DHT segment into multiple segments which collectively hold the exact same information as the original single segment Even though the contained data has not changed, the simple act of splitting it will change the hash value for the file. The encoded image data is not held in a segment as there is no restriction on its length. The encoded data is assumed to start immediately after the SOS segment and finish immediately before the EOI marker. It should also be noted that as well as the main image, a JPEG usually contains a thumbnail nested inside the metadata section which will be stored as separate JPEG with its own SOI, compression information segments, encoded data and EOI; it will not usually contain any metadata.

3.1.1. JPEG metadata segments

Sixteen *application segments* are allocated for metadata, referred to as APP0. . . APP15 using the markers 0xFFE0 to 0xFFEFc [29]. An application segment can be used for any purpose and multiple instances of a particular segnment can be used within a file; the specific purpose of each segment is indicated by a signature immediately following the two bytes stating segment size. Metadada readers should skip over a segment if they do not recognise the signature. The most common segments apart from APP0 and APP1 are:

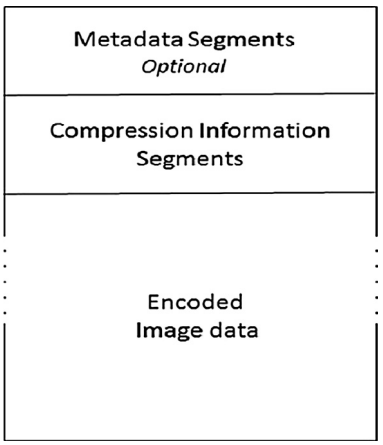

Fig. 2. JPEG Overall Structure.

APP2: Originally dedicated to Flashpix tags. Introduced in 1996, the Flashpix format is now obsolete but some of its structures are included in the EXIF specification and may still be encountered in some cameras; also used by Photoshop when editing a camera image.
APP13: used by Adobe for Photoshop tags.
APP14: used by Adobe for image encoding information relating to DCT filters.

APP0 must be used for JFIF information where it is included, signed with the ASCII values for the character "JFIF". It is used by software applications for compatibility with other applications and does not appear in an image straight from camera. It is therefore of no direct interest to this project except in regard to taking account of its possible presence when analysing or processing a file. Strictly speaking, the JFIF and EXIF specifications are incompatible because both of the specifications state that the segment containing their data should be the first segment in the JPEG file. In practice, they coexist without any problems and when a JPEG file is created in a digital camera and later edited in a software application, it will typically contain JFIF data in its first segment marked as APP0 and EXIF data in its second segment marked as APP1. APP1 is used by both EXIF and XMP for metadata and there will be two APP1 Segments if the file contains both EXIF and XMP. The signature for an EXIF segment is the ASCII values for "Exif" followed by two nulls. Fig. 3 shows the EXIF signature highlighted.

The signature for an XMP segment is the ASCII values for http://ns.adobe.com/xap/1.0/\x00; an example is shown in Fig. 4. Analysis of files using a hex editor is excessively laborious and tedious so we developed JPEGReader, an application to view JPEG structures in a much simpler way. A different version, PNGReader, was also developed for reading and modifying PNG files. The following screenshots from JPEGReader illustrate the different ways in which segments can be used inside a JPEG file.

Fig. 5 shows the segments structure for an image file straight from a digital camera. The file contains a single APP1 segment holding EXIF data, some of which can be seen in the text values. The markers combo box. JPEGReader also provides some details about the segment such as its start address in the file and its length. The APP1 segment is followed by five segments relating to the decompression – DQT, DRI, SOF0, DHT and SOS. The start of encoded data is indicated by 0000 () as encoded data is not in a segment and therefore has no segment marker. Fig. 6 shows

Fig. 3. EXIF Signature.

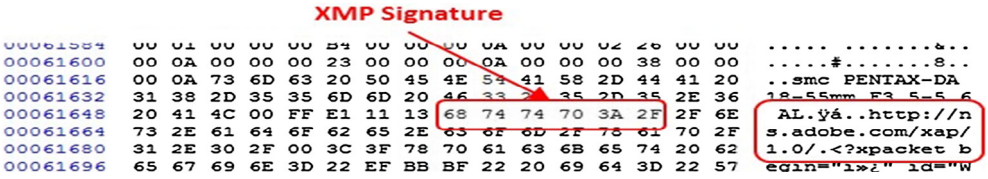

Fig. 4. XMP Signature.

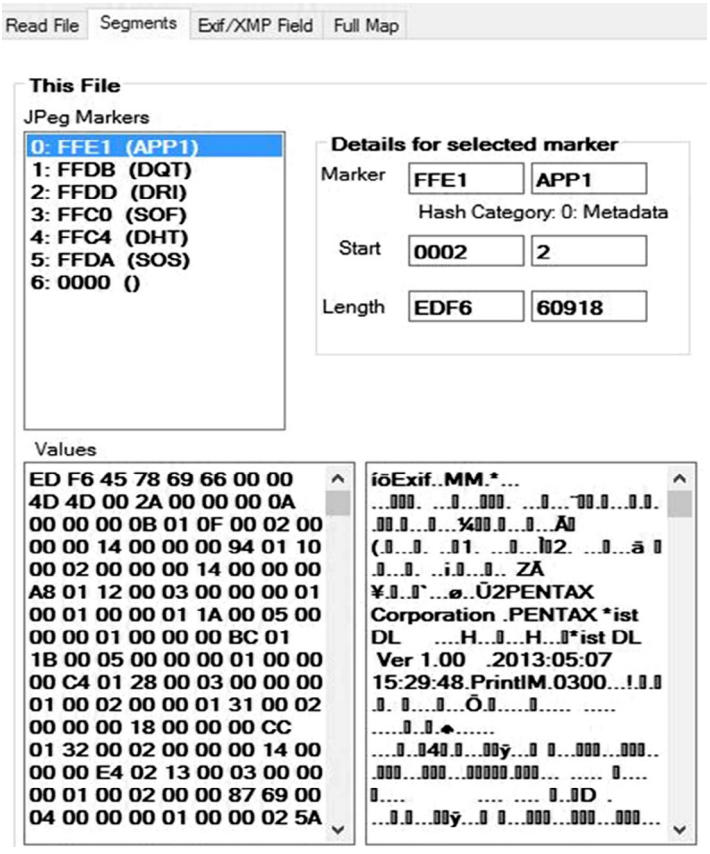

Fig. 5. JPEG straight from camera.

the structure after adding some metadata through Windows Explorer. A second APP1 segment has been added to hold XMP data. The XMP signature can be seen in the values box below the markers combo box. Windows Explorer has split the DQT data into two segments and the DHT data into four segments. There has been no overall change to the data and despite extensive research, it has not been possible to find an explanation for why Windows splits these segments.

Fig. 7 shows the structure for the original image with the same metadata added using Bridge, an Adobe application for image management (but not image processing). Like Windows Explorer, Bridge adds a second APP1 to hold XMP. For some unknown reason, it also adds an APP13 segment holding some Photoshop style metadata Photoshop related data, even though this particular image has not been processed in Photoshop and that same information is already stored in the XMP segment. Unlike Windows Explorer, the segments containing compression related data are left as single segments (see Fig. 8).

Fig. 7 shows the effect of opening the original file in Photoshop and re-saving it. Although no changes have been made to the image, it is treated as having been processed by software so an APP0 segment is added to the start to hold JFIF data. As well as the second APP1 and the APP13 segments added by the previous example in Bridge, an additional APP14 segment is added to hold additional Photoshop data.

At this stage of the research, it was clear that application segments offer a great range of flexibility and it seemed that an application segment could be an ideal place for holding a digital

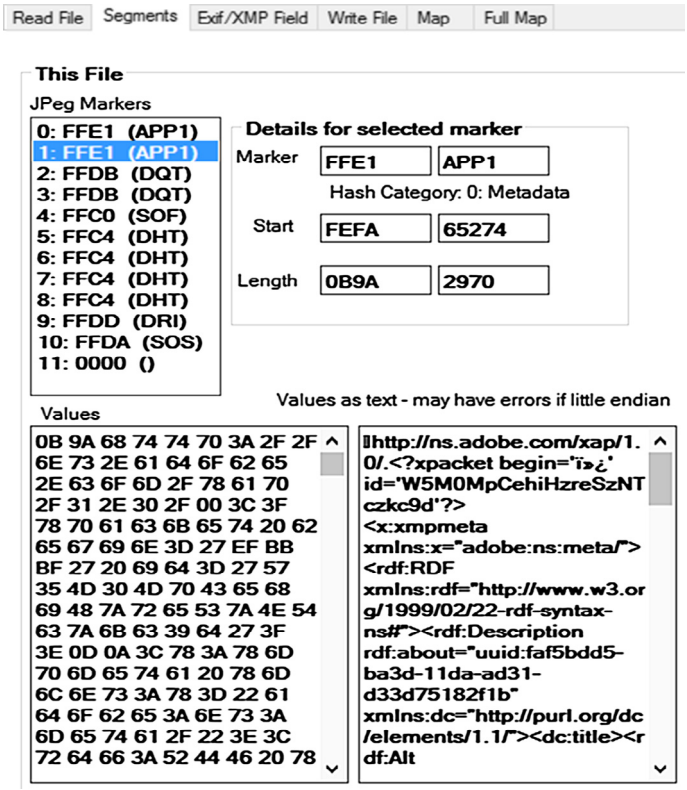

Fig. 6. JPEG edited in Windows Explorer.

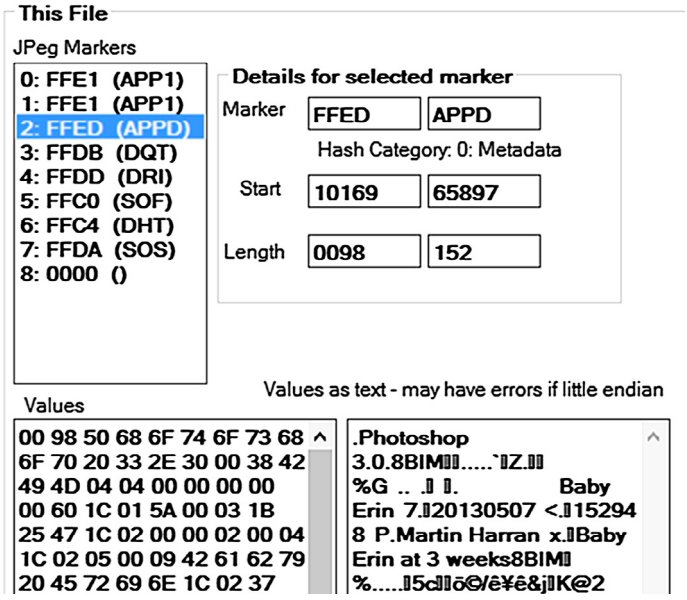

Fig. 7. JPEG edited in Adobe Bridge.

certificate. The potential for this was confirmed by a 'quick and dirty' test, manually inserting an existing certificate into the original image used in the previous examples. The results are shown in Fig. 9. The digital certificate has been inserted into an APP1 segment and the text box shows the text to which the digital certificate applies. Initially, it was thought that a less common application segment such as APP8 should be used but it was found that anything other than APP1, APP13 or APP14 could, in some circumstances, be deleted by the iOS platform which is the target of our mobile application. APP1 was found to be generally safe from overwriting.

3.2. EXIF metadata structure

EXIF metadata is always contained in an APP1 segment which is specified to be the first segment in the file but can in practice be preceded by an APP0 segment. The presence of EXIF metadata is indicated by the EXIF signature immediately after the two bytes

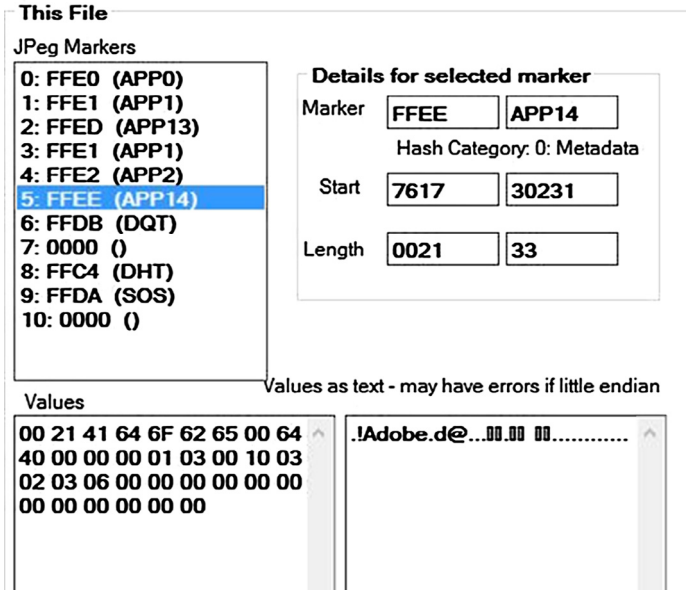

Fig. 8. JPEG edited in Adobe Photoshop.

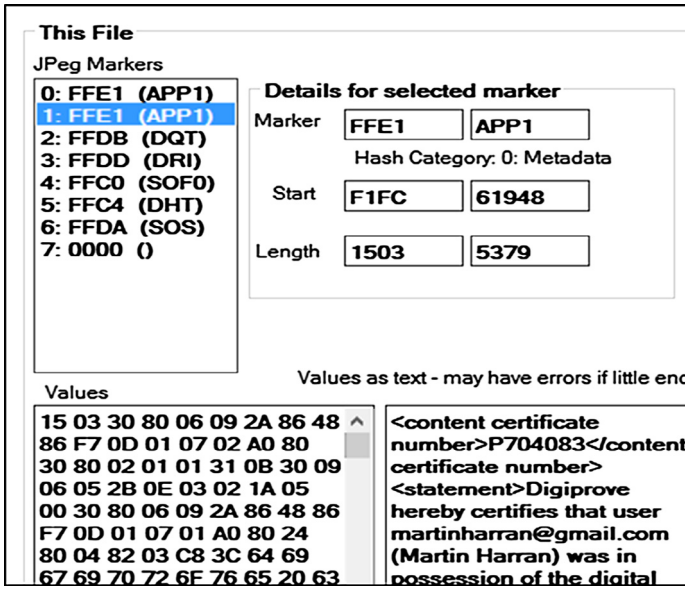

Fig. 9. JPEG file with certificate inserted.

stating the segment size. That signature is the ASCII values for "Exif" followed by two nulls. The EXIF structure is based on the TIFF specification and the EXIF signature is immediately followed by an 8-byte TIFF header. An example of a TIFF header is shown in Fig. 10. The format is as follows (hex values shown in italics are from Fig. 10) (see Fig. 11).

Bytes 0–1: *0x4D4D.* Byte order within the TIFF segment. There are only two legal values – 0x4949 ("II") indicating little endian and 0x4D4D ("MM") indicating big endian. The letters "I" and "M" are from Intel and Motorola, the two dominant chipset manufacturers at the time TIFF was developed; Intel used a little endian architecture and Motorola a big endian one. TIFF endianness varies among camera manufacturers – Canon uses little endian, Pentax use big endian – and among mobile phone manufacturers - Samsung uses little endian while Apple use big endian. Byte 0 also has an important role for the calculation of offsets; TIFF makes extensive use of these offsets and they are all calculated from the address of byte 0.

Bytes 2–3: *0x002A.* An arbitrary but carefully chosen number (42) that further identifies the file as a TIFF. Effectively, the first two endianness bytes along with these two give TIFF a unique signature.

Bytes 4–7: *0x0000000A.* The offset (in bytes) of the first Image File Directory (IFD) which holds information about the image as well as pointers to the actual image data. The value 0x0000000A in this case added to the start address of the TIFF header – 0x000C – tells us that the first Image File Directory starts at 0x0016.

An Image File Directory has a structure where the first two bytes state the number of fields that the IFD contains. The final 4 bytes of the IFD give the offset to the next IFD from the start of the TIFF header; they are set to zero if this is the final IFD.

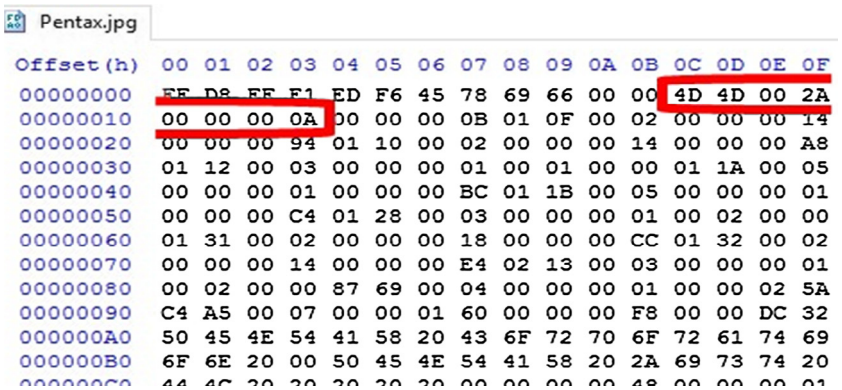

Fig. 10. TIFF Signature.

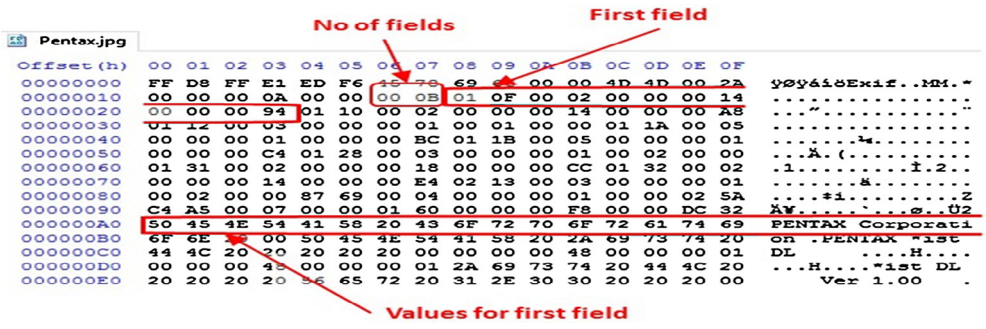

Fig. 11. Image File Directory.

Fig. 12 shows the physical file layout for a range of devices including three digital cameras from leading manufacturers, an iPhone, a Windows Phone, and two Android Phones - one high end and one low end. Several things are worth noting in Fig. 12.

I. The sequence of IFDs varies between devices; Pentax and Canon, for example put Interop IFD before IFD1 which comes last in the sequence but Nikon put the Interop IFD last.
II. A number of the devices do not include an Interop IFD at all even though this is mandated in the DCF which they are all supposed to follow. The default values for the Interops tags, however, are "R98" in the first tag which is a reference to the 1998 DCF that first introduced the Interops IFD and Version 1.0 in the second tag. Closer analysis showed that all the devices that include the IDF have those default values; whilst documentation of this could not be found, it seems reasonable to conclude that devices that do not specifically include the IFD work are in fact using those defaults.
III. Although all the devices have chosen to place values for each IFD immediately following the IFD, that is not strictly necessary as will be seen later when editing metadata is discussed.

3.3. Metadata tags

IFD0 tags are not included as there are 231 of them but only a handful are generally used in JPEG. Although there are no rules in the EXIF specification about the order of IFDs, all tags inside an IFD must be in numerical order. The first 10 tags shown (F0. . .F9) are generally used in IFD0 as well as the Exif and GPS tags which can be seen at the bottom of the list of tags in this. There are 4 tags identified as unknown. No documentation could be found for those tags, they are not listed in the TIFF or EXIF specifications and their values are all set to zero, so they appear to be private ones added by Huawei but either not used or reserved for future use. Hauwei, however, appear to have broken TIFF rules here – all private tags used in a TIFF IFD are mandated to use id markers above 0x8000 [30] as can be seen, for example, with the Exif and GPS private tags. Examples of EXIF and GPS tags in the same file are shown in Fig. 13.

There are over 30 GPS tags available; Latitude, Longitude and Altitude tags are ubiquitous; other GPS tags vary between devices. IFD1 holds the image thumbnail and can, in theory, contain any TIFF tags i.e. the same as IDF0. In practice, however, only the tags in Fig. 14 are generally used. The tags for ThumbnailOffset (0x0201) and ThumbnailLength (0x0201) are used only in IFD1, having been deprecated for IFD0 where the image data is taken to start immediately after the SOS segment and run to the end of the file, ending with 0xFFD9. The tag MakerNote (0x927C) is stored inside the EXIF IFD and is another tag that is used to point to data stored in an IFD type structure. It is intended for free use by the camera or device manufacturer and generally contains extended data about the camera, the lens and the various settings for when the image was captured. Not all manufacturers use MakerNote, Hauwei for instance do not and there are major inconsistencies among manufacturers who do include it. First of all, manufacturers use different lengths for the initial string of bytes that usually act as a signature for the manufacturer. Secondly, whilst the actual metadata is stored in an IFD structure, some manufacturers include an internal TIFF header and this becomes the reference point for calculating offsets to the tag values. Other manufacturers calculate offsets relevant to the TIFF header at the start of the APP1 segment; that means that if the tag values get moved (see next section) then the offsets have to be all recalculated or they will be wrong. In practice, that recalculation is made extremely difficult if not impossible due to very little if any information about MakerNote

Digital Camera			Mobile Phone			
Pentax ist DL	**Nikon E2500**	**Canon PowerShot A80**	**iPhone 4S (iOS)**	**Nokia Lumia 625 (Win 8)**	**Samsung SM-N9005 (Android)**	**Huawei Y360 (Android)**
Big Endian	Little Endian	Little Endian	Big Endian	Big Endian	Little Endian	Little Endian
IFD0	IFD0	IFD0	IFD0	IFD0	IFD0	IFD0
IFD0 Values	IFD0 Values	IFD0 Values	IFD0 Values	IFD0 Values	IFD0 Values	IFD0 Values
Exif IFD	Exif IFD	Exif IFD	Exif IFD	Exif IFD	Exif IFD	Exif IFD
Exif Values	Exif Values	Exif Values	Exif Values	Exif Values	Exif Values	Exif Values
Interop IFD	IFD1	Interop IFD	GPS IFD	GPS IFD	Interop IFD	GPS IFD
IFD1	IFD1 Values	IFD1	GPS Values	GPS Values	GPS IFD	GPS Values
IFD1 Values	Interop IFD	IFD1 Values	IFD1	IFD1	GPS Values	IFD1
			IFD1 Values	IFD1 Values	IFD1	IFD1 Values

Fig. 12. JPEG file layout for different devices.

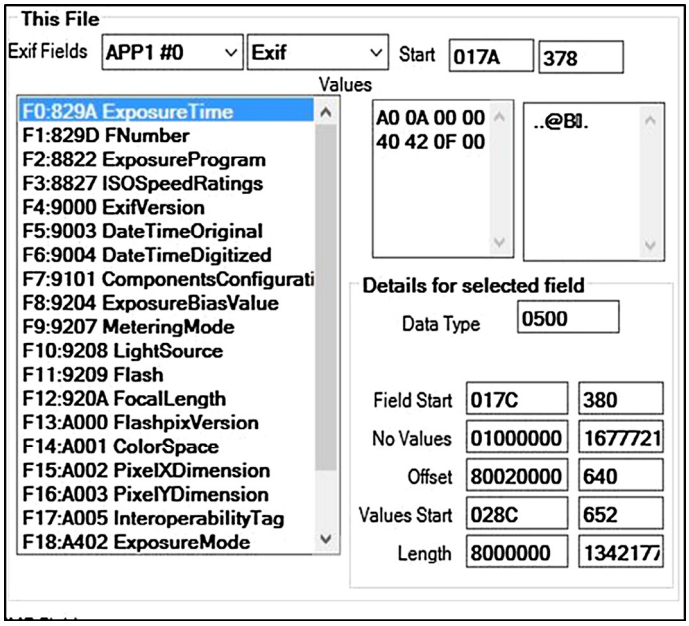

Fig. 13. Tags in EXIF IFD.

being published by the manufacturers; most of the information that is available has come from reverse engineering by researchers like Phil Harvey, developer of ExifTool [31] which reads and writes metadata for a vast range of file formats, executables as well as image, audio and video files. His website provides the most comprehensive list available of information relating to metadata across these formats including all known JPEG, TIFF and EXIF tags [32]. Leading camera manufacturer Canon includes over 80 tags in MakerNote identified by ExifTool.

4. Evaluation

Changes in metadata do not necessarily affect the integrity of a file if the core data of the file is not changed. This is no different from paper documents where notations added to a document do not affect the validity of the document. If a person receives an image file from someone else and, for example, adds a tag to help find it afterwards then that should not affect the integrity of the actual image. That principle should equally apply to metadata changes arising from the transfer of a file between devices or operating systems which should not be invalidated so long as the image data is not altered. This should also apply to the compression segments in a JPEG file which essentially are metadata about the compression used; for example, if a single DHT segment is split into four that should not affect the validity of the file again as long as the encoded image data is not affected. We propose that the primary fingerprint for a JPEG file should be the hash value for the encoded image data. This is analogous

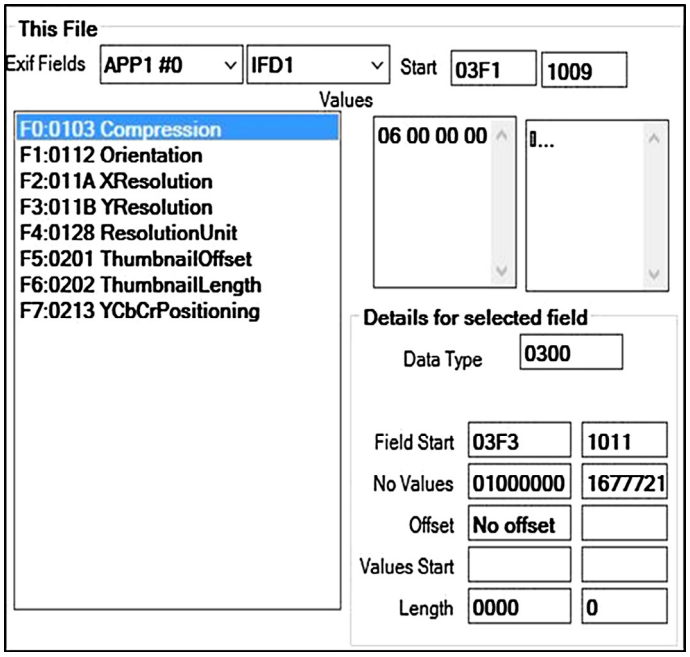

Fig. 14. Tags in IFD1.

to the approach taken by Microsoft Word and Adobe Acrobat which concatenate a digital signature with the document to which it relates and, when the document is being verified, the digital certificate is ignored for computing the hash value of the document.

We also retain three fingerprints - metadata, compression and image data which we developed in JPEGReader as it will allow us to build into its visibility approach a green/amber/red symbolism where green means a file has been unaltered since the digital certificate was issued, amber means that metadata or compression has been altered but the core image data is not changed so the digital certificate is still valid and red means that the digital certificate is no longer valid as the core image data has been altered. A key requirement is the ability to insert additional data as well as a digital certificate. For example, the comment metadata field should hold a URL to a website where a user can check the validity of a file without having third party software installed by users. We also incorporate automatic copyright stamping. In order to retain maximum flexibility, Title, Author, Copyright and Comments are enumerated fields within our software but additional fields could be inserted by supplying the appropriate tag index along with the values.

Changes can take place in metadata either deliberately by a user or inadvertently as a result of moving the file between devices and operating systems. It would therefore be safest to insert and digital certificate into its own application segment. iOS has intolerance for application segments other than APP1, APP13 and APP 14 in images received by email. Like other applications, however, it does allow multiple instance of APP1. It was therefore decided to use an APP1 segment to hold the digital certificate. This APP1 would be distinguished from other APP1s by its own signature. An example of a JPEG with a digital certificate inserted is shown in Fig. 15. The first APP1 is selected on the left and the EXIF fields can be seen in the values; the second APP1 is selected on the right and the signature can be seen in the values along with the initial part of the certificate. The overall algorithmic approach for creating a file with references and a digital certificate inserted can be summarised as:

	Overall Process	External Application	This Library
1	Open file	X	
2	Get reference	X	
3	Insert reference & accompanying data into metadata		X
4	Calculate SHA256 hash value for metadata segments.		X
5	Calculate SHA256 hash value for compression segments		X
6	Calculate SHA256 hash value for image data.		X
7	Return hash values as pointer to array of pointers.		X
8	Return hash values to owner	X	
9	Get digital certificate	X	
10	Insert digital certificate into own segment.		X
11	Return file as pointer to new data stream.		X
12	Save and close file.	X	

The corresponding process to verify a file is straightforward:

1 Extract digital certificate from file.
2 Read hash values from digital certificate.
3 Compute hash values for each section of the file - metadata segments, compression segments and encoded image data.
4 Compare computed hash values to the values extracted from the digital certificate and report as appropriate:
0 = the file has been unaltered since the digital certificate was issued.
1 = the metadata or compression has been altered but the core image data is not changed so the digital certificate is still valid.
2 = the digital certificate is no longer valid as the core image data has been altered

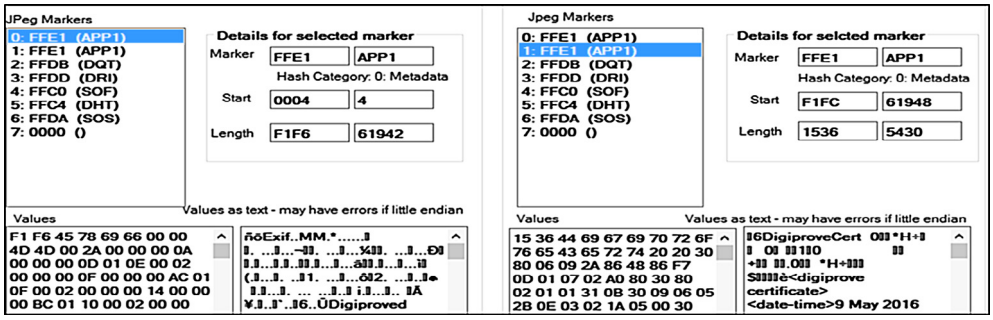

Fig. 15. Digital certificate in an APP1.

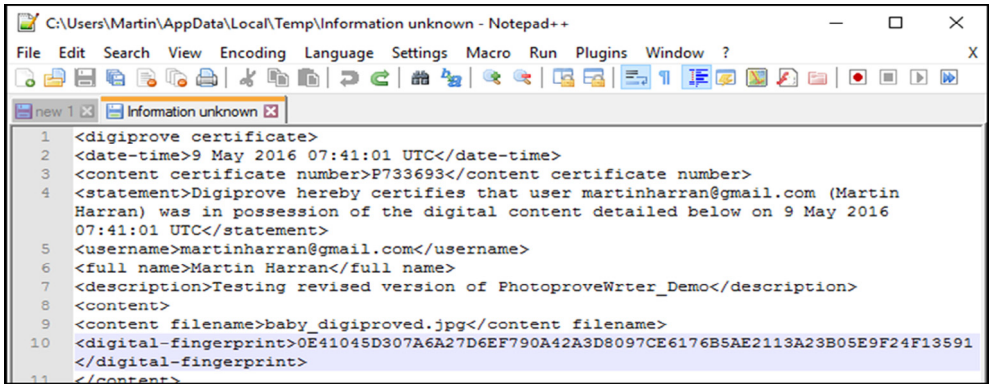

Fig. 16. Testing - view original text.

We developed JPEGReader and PNGReader to assist in the analysis of image files. These were developed in C# using Microsoft Visual Studio. In order to keep the header file uncluttered, the code was written used the *Pimpl Idiom*, also known as the *Cheshire Cat* or *Opaque Pointer* which is a method of avoiding the forward declaration of all private variables and methods in a C++ header file [33]. Our documented code is available at http://www.api.masters.martinharran.com. Our work shows that it is possible to insert a digital certificate into a JPEG file and to edit the EXIF metadata fields to show references to a certificate. These would be visible in native applications such as Windows Explorer and OS X Inspector as well as third party image viewers and editors.

Independent testing was carried out by Georgia Tech out on the original Digiprove process.[3] The procedure used by Georgia Tech was replicated for the process developed in this project:

Step 1: For selected file samples, hash values were precalculated using the same encryption used by us (software used was sha256deep from md5deep version 3.4).
Step 2: Each file was then put through our content signing process to generate a digitally-signed content certificate.
Step 3: A 3rd party digital signature verification utility (Certificates, Microsoft Corporation, Version: 5.1.2600.0) was used to verify the integrity of the signature using our public X.509 digital identity certificate (issued by VeriSign).

Test metadata was inserted into an image file (baby.jpg) using the Photoprove writer library. The SHA256 value for this file was calculated using sha256deep from md5deep v4.3. The result: c7a0a355a9b5926f7c332807e4ca380e378e20aee652a1f6d066276aab6324fe was then put through a content signing process which yielded the same SHA256 value as Step 1. The resulting PS7 file produced and emailed was inserted into an APP1 using the Photoprove writer library. In order to confirm persistence, the file was copied across devices and operating systems in a similar process to that used earlier for testing persistence across operating systems i.e. Windows via USB to Linux via DropBox to OS X via iCloud to iPhone via Windows Explorer to Windows 10 (baby _final.jpg). A hex editor – HxD – was then used to manually strip the digital certificate back out of the final file and save it as a PS7 file (CertExtracted.p7s) and the JPEG minus the stripped APP1 was resaved (baby_final_cert_extracted.jpg). This allowed the processed file to be verifoied against the extracted digital certificate. SHA256 was recalculated for the stripped file and was identical. The PS7 file extracted at Step 2 was opened in PS7Viewer, which verified the digital signature. P7S viewer also has the facility to open the original unsigned text to which the digital signature applies; this is shown in Fig. 16 where the file fingerprint can be seen (line 10) along with the date/time stamp and other information inserted. Again, that fingerprint matches the one calculated in Step 2 using sha256Deep.

We next checked the effect of sending the file with the inserted certificate to an iPhone by email and comparing the original with the image saved by the iPhone. As it was already known from the previous testing that the iPhone would change the metadata by creating a new thumbnail, there was no point in comparing the overall hash value but Fig. 17 shows that the second APP1 holding the digital certificate has remained intact and Fig. 18 shows that only the hash value for the metadata section has changed, the compression segments and image data are unchanged.

[3] *Report for process evaluation & penetration testing service (D9507)* Submitted by: Mayur Ramgir, Georgia Tech Ireland, Athlone Business & Technology Park, Garrycastle, Dublin Road, Athlone mayur.ramgir@gtri.gatech.edu.

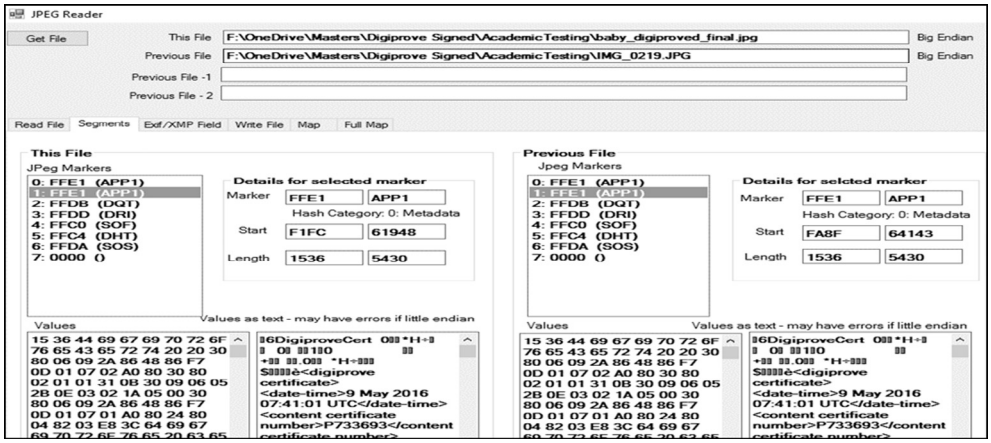

Fig. 17. Testing email to iPhone - Digital Certificate.

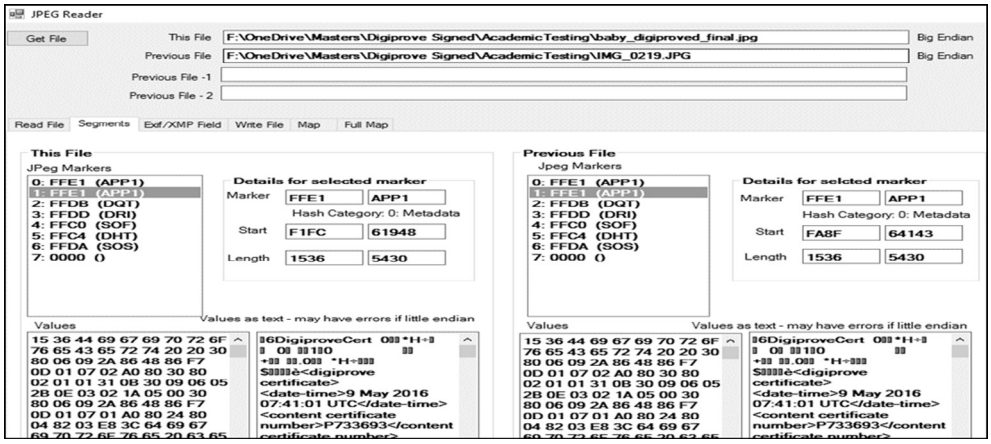

Fig. 18. Testing email - hash values.

5. Conclusion

We demonstrate that a digital certificate relating to an image file can be inserted inside that image file along with accompanying metadata containing references to the issuing company. Notwithstanding variations between devices and across operating systems and applications, a JPEG file holds its structure very well. Where changes do take place, this is generally in the metadata area and does not affect the encoded image data which is the heart of the file and the part that needs to be verifiable. References to the issuing company can be inserted into the metadata for the file. There is an advantage of having the digital certificate as an integral part of the file to which it applies and consequently travelling with the file. We ultimately prove that the metadata within a file offers the potential to include data that can be used to prove integrity, authenticity and provenance of the digital content within the file.

Our research has also uncovered a wide range of previously unknown aspects of how metadata gets handled by various operating systems, devices and image management applications. This is knowledge that could be valuable to others working in the field. An unanticipated benefit of the work has been the software tools developed for examining and analysing image files. Those tools have already attracted the interest of the wider forensics community. Due to their massive popularity, image files, especially JPEG, offer high high potential as carriers of other information. Much of the work to date on this has focused on stenographic ways of hiding information using least significant bit techniques but we believe that the findings in this project have exposed other ways of doing this. If a digital certificate can be inserted into a JPEG, so could the payload for a virus or a communication between criminal or terrorist organisations.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.aci.2017.05.006.

References

[1] F. Buccafurri, L. Fotia, G. Lax, Allowing privacy-preserving analysis of social network likes, in: Eleventh Annual International Conference On Privacy, Security And Trust, Pst 2013, Eleventh Annual International Conference on Privacy, Security and Trust, PST 2013, IEEE Computer Society, Tarragona (Spain) (USA), 2013.

[2] F. Buccafurri, L. Fotia, G. Lax, Allowing Non-identifying Information Disclosure in Citizen Opinion Evaluation, Egovis/edem 2013, August 26–28, 2013, in: Proceedings of International Conference on Electronic Government and the Information Systems Perspective and International Conference on Electronic Democracy (EGOVIS and EDEM 2013), Springer Verlag, Prague, Czech Republic (DEU), 2013, pp. 241–254.

[3] K.N. Sowmya, H.R. Chennamma, Video authentication using watermark and digital signature—a study, in: S. Satapathy, V. Prasad, B. Rani, S. Udgata, K. Raju (Eds.), Proceedings of the First International Conference on Computational Intelligence and Informatics Advances in Intelligent Systems and Computing, vol. 507, Springer, Singapore, 2017.

[4] F. Buccafurri, L. Fotia, G. Lax, Allowing Continuous Evaluation of Citizen Opinions Through Social Networks. International Conference on Electronic Government and The Information Systems Perspective and Intern, in:

Proceedings of International Conference on Electronic Government and the Information Systems Perspective and International Conference on Electronic Democracy (EGOVIS and EDEM 2012), Springer Verlag, Vienna (DEU), 2012.
[5] F. Buccafurri, L. Fotia, G. Lax, Privacy-Preserving Resource Evaluation in Social Networks, Tenth Annual Conference on Privacy, Security and Trust (pst 2012), in: Proceedings of the Tenth Annual Conference on Privacy, Security and Trust, IEEE Computer Society, Paris (USA), 2012, pp. 51–58.
[6] Y. Ye, K. Xie, Q. Zeng, A study on DITA in digital publishing, in: Proc. SPIE 10322, Seventh International Conference on Electronics and Information Engineering, 103224N (January 23, 2017); http://dx.doi.org/10.1117/12.2265276.
[7] C. Kinsella, Establishing Proof of Existence and Possession of Digital Content, Irish Patent Office 84803, 2008. <https://eregister.patentsoffice.ie/HttpHandler/Handler.ashx?HandlerType=PDF&DocumentType=PT&DocumentId=22647205>.
[8] Microsoft Office Support. Digital Signatures and Certificates, Office Support, 2016. <https://support.office.com/en-us/article/Digital-signatures-and-certificates-8186cd15-e7ac-4a16-8597-22bd163e8e96>.
[9] P. Christof, P. Jan, Understanding Cryptography: A Textbook for Students and Practitioners, Springer, Berlin, Heidelberg, 2010.
[10] A.J. Menezes, P.C. Van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996.
[11] P. Rogaway, T. Shrimpton, Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance, Fast Software Encryption (2004) 371–388 (Springer).
[12] R. Rivest, RFC 1321: The MD5 message-digest algorithm, April 1992', Status: INFORMATIONAl, 1992.
[13] H. Dobbertin, The status of MD5 after a recent attack, CryptoBytes 2 (2) (1996).
[14] X. Wang, Y.L. Yin, H. Yu, Finding collisions in the full SHA-1, in: Advances in Cryptology–CRYPTO 2005, Springer, 2005, pp. 17–36.
[15] A. Sotirov, M. Stevens, J. Appelbaum, A.K. Lenstra, D. Molnar, D.A. Osvik, B. de Weger, MD5 considered harmful today, creating a rogue CA certificate', in: 25th Annual Chaos Communication Congress, 2008.
[16] C.R. Dougherty, Vulnerability Note VU# 836068 MD5 Vulnerable To Collision Attacks', 2008. <https://www.kb.cert.org/vuls/id/836068>.
[17] Swiat. Flame Malware Collision Attack Explained, Microsoft Technet, 2012. Available: <https://blogs.technet.microsoft.com/srd/2012/06/06/flame-malware-collision-attack-explained/>.
[18] R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM 21 (2) (1978) 120–126.
[19] NIST. 'Secure Hash Standard (SHS)', FIPS PUB 180-1, 1995. Available: <http://www.umich.edu/~x509/ssleay/fip180/fip180-1.htm>.
[20] S. Gueron, S. Johnson, J. Walker, SHA-512/256, Information Technology: New Generations (ITNG), in: 2011 Eighth International Conference on, IEEE, 2011, pp. 354–358.
[21] X. Wang, D. Feng, X. Lai, H. Yu, Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, IACR Cryptology ePrint Archive, 2004, 199.
[22] NIST, 'Secure Hash Standard (SHS)', FIPS PUB 180-2, 2002. Available: <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>.
[23] NIST. 'SHA3 Standard: Permutation-based hash and extendable-output functions', FIPS PUB 202, 2015. Available: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>.
[24] NIST, NIST'S POLICY ON HASH FUNCTIONS, National Institute of Standards and Technology, 2016. Available: <http://csrc.nist.gov/groups/ST/hash/policy.html>.
[25] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Trans. Inform. Theory 22 (6) (1976) 644–654.
[26] N. Leavitt, Internet security under attack: the undermining of digital certificates, Computer 44 (12) (2011) 17–20.
[27] M. Fioretti, How to Add Metadata to Digital Pictures from the Command Line, Linux.com, 2008. Available: <https://www.linux.com/news/how-add-metadata-digital-pictures-command-line>.
[28] J.S. Moreland, Dream.In.Code -> Updating Jpeg Metadata Wthout Loss of Quality [online], Dream In Code, 2010. Available: <http://www.dreamincode.net/forums/blog/1017/entry-2305-updating-jpeg-metadata-wthout-loss-of-quality/>.
[29] P. Harvey, JPEG Tags, ExifTool, 2016. <http://www.sno.phy.queensu.ca/~phil/exiftool/TagNames/JPEG.html>.
[30] IPTC, Embedded Metadata Initiative - Manifesto [online], Embedded Metadata Initiative, 2016. <http://www.embeddedmetadata.org/embedded-metatdata-manifesto.php>.
[31] P. Harvey, ExifTool by Phil Harvey, ExifTool, 2016. Available: <http://www.sno.phy.queensu.ca/~phil/exiftool/>.
[32] P. Harvey, ExifTool Tag Names, ExifTool, 2016. <http://www.sno.phy.queensu.ca/~phil/exiftool/TagNames/index.html>.
[33] Z. Miners, Facebook, Twitter Called out for Deleting Photo Metadata [online], Network World, 2013. Available: <http://www.networkworld.com/article/2164374/software/facebook–twitter-called-out-for-deleting-photo-metadata.html>.