A New Technique for Authenticating Content in Evolving Marked-up Documents

Phillip Berrie
University of New South Wales at ADFA, Australia

Abstract

Accuracy of transcription is vital when preparing a scholarly version of an existing document. This process has not changed with the advent of electronic editions. In fact, ensuring the continued accuracy of a transcription in the digital realm is more difficult because a file, unlike a piece of paper, does not retain information about its previous states and it is therefore possible that accidental changes can go undetected unless the content is continually checked against the original.

This article presents a new, character-set-independent, programming algorithm that allows for the ongoing authentication of the textual content of files being marked up with SGML-like languages. The study also describes an implementation of this algorithm and how it can be used with existing software tools to provide a more efficient and trusted editing environment for creating and editing marked-up files.

The Just In Time Authentication Mechanism (JITAM) algorithm was developed in response to the need for some form of automated authentication mechanism for projects already employing embedded markup and is seen as a preparatory step that editors can take with their projects before making the leap to the more versatile Just In Time Markup (JITM) system.

Correspondence:
Phillip Berrie,
Australian Scholarly Editions Centre,
School of Humanities and Social Sciences,
University of New South Wales at ADFA,
Canberra ACT 2600,
Australia.
E-mail:
p.berrie@adfa.edu.au

1 Introduction

The Just In Time Authentication Mechanism (JITAM) algorithm is the most recent development of the Electronic Collaboration in Humanities Research projects at the Australian Scholarly Editions Centre (ASEC) at the University of New South Wales at the Australian Defence Force Academy in Canberra.[1] JITAM is a logical extension of the Just In Time Markup (JITM) system, reported at the Computing Arts conference in Sydney 2001 (Barwell *et al.*, 2003). Many of the ASEC's online projects use JITM system to dynamically generate user-customized versions of texts for display through a standard web browser. Therefore, to assist in providing a context for a more formal description of the JITAM algorithm, a brief description of the JITM system is warranted.

Instead of maintaining transcription and markup for a text in the same file, as is the case in most electronic editions, a JITM system maintains a minimally marked-up base transcription file and a number of separate tagset files.[2] The JITM system uses commonly available, non-proprietary, platform-independent, web-based CGI technologies to bring these selected component parts together to create a user-specified 'Perspective' on the text. The same software can run on both a desktop machine and an internet accessible web server.

The tagset files hold the data required for selectively embedding markup into the text. The abstraction of the subjective and interpretative

Literary and Linguistic Computing, Vol. 22, No. 1, 2007.
doi:10.1093/llc/fqi047 Advance Access Published on 3 November 2005

markup away from the transcription and its subsequent categorization allows for a large number of possible combinations of the data, thus catering for a wide range of interests. This data abstraction concept also provides support not only for the continued development of the text as new tagsets are created and added to the system, but also enables the creation of Perspectives of alternative points of views, both structural and annotative, of the same text. Furthermore, it can reduce complexity by allowing for the omission of unwanted markup when the JITM system brings the user-specified components together in a Perspective.

The JITM system has, as one of the cornerstones of its operation, the automatic authentication of its base transcription files. To avoid misrepresentation of the text, the system reports an error instead of an incorrect reading if authentication fails. The tagset files that contain the commands for embedding markup are an integral part of this authentication mechanism, and each of the markup commands in the tagset files contains the means of authenticating the text element to which it is to be applied.

This concept gives the JITM system another important strength: the development of markup for a text can be done simultaneously at multiple sites and work of different sites can be combined to augment the same text. The transfer of data between sites also provides an integrity-checking mechanism between sites.[3] Further detail on the JITM system is beyond the scope of this article: more information can be found at the following web site: http://www.unsw.adfa.edu.au/ASEC/JITM/Publications.html

Being able to trust the stability and authenticity of transcribed texts is a major concern for scholars in the humanities. The majority of authentication tools available today are based on the premise of a static file, and very little is available for the documents that are undergoing development. Yet this is a common situation in the humanities; transcription of a historical text from a physical document with interpretation added to it later in the form of an additional markup. The JITM system neatly deals with this problem, but it has not been adopted by scholars in the humanities as quickly as expected.

One of the disadvantages with developing a new paradigm is that, as well as changing people's minds, one has to overcome the inertia of the existing paradigm. It became clear that the amount of intellectual property already invested in projects using complicated all-in-one marked-up files was going to prevent a lot of people changing the way they worked—even though they could see the obvious advantages of the JITM system and its authentication mechanism.[4]

The algorithm described here, and the software developed from it, are freely available to the academic community as an aid to the creation and maintenance of marked-up files, whenever ensuring the accuracy of the content is important. Since the implementation of this algorithm will make the user's files amenable to future implementation in a JITM system, it is hoped that this will assist in the migration of practitioners away from the inefficient practices of the current paradigm.

2 The Rationale for the Algorithm

There are two basic requirements for the JITAM algorithm to work. The first is that the basic textual content—the lexical text—of the files be static (as in the case of transcription from physical originals). The second is that this content be marked up using SGML-like markup languages, XML-based in preference.

The design of the algorithm is premised on the need for platform independence and the dichotomous structure of markup files.

2.1 The need for platform independence

It is a sorry fact that data transfer between computers is still not as easy and transparent as it should be due to the proliferation of different operating systems, file formats, and character sets. Even with SGML-like languages, where the file encoding is in the human readable ISO-646 character set, there are problems brought about by different platforms handling ends-of-lines differently and using different character encodings (e.g. ASCII, UTF-8, UTF-16).[5]

Current authentication schemes (e.g. MD5[6]) cannot handle the authentication of files that are transferred and edited between platforms. For example, text files created on a Macintosh and a Windows machine that contain the same visible characters do not generate the same MD5 checksum.[7] This is because of the different ways of identifying an end-of-line on these two computing platforms.

Similarly, the push by computer vendors to make their products globally acceptable has brought Unicode (or ISO-10646) and its variant encodings upon us. Considering the rate at which old technologies become obsolete, how long will it be before ISO-646 (or ASCII) files become unreadable by computers? And creating new Unicode versions of files is not as simple as it sounds because, unless the files are proofread, character-by-character, it cannot be proven that the translation has occurred accurately. This is because the current means of automatically checking file integrity cannot work if the file is actively being changed.

Byte-oriented authentication schemes, like MD5, cannot be used in the two cases mentioned above as they use the bit-wise numerical value of a byte in the calculation of their checksum.[8] Traditionally, each byte represents a character, but not in Unicode or its variants. In Unicode character-sets, a character can be represented by 1 (i.e. UTF-8), 2 (i.e. UTF-16), 4 (i.e. UTF-32) or more bytes. Therefore, this byte-oriented technology can only provide protection within the domain of its own character set. Different platforms have different character sets and, to make things worse, the order in which the bits are stored in a byte can differ.[9]

The algorithm being proposed here handles this problem by basing its arithmetic on the offset of a character from the character-set-specific numerical value of a base character. This idea builds on the acceptance of the common ordering of the characters used in computer character sets, which is typically a superset of alphabetical ordering.[10] If this normal order for characters is used then the offset value will always be the same no matter what the numerical values for the characters all. For example, the numerical value for the letter 'f' should always be five greater than the numerical value of the letter 'a' no matter what the numerical value of 'a' is for the character set in question.

For most cases, this is a non-problem because the developers of Unicode have treated ASCII as a subset of the Unicode character set. However, by dealing with the contents of a file as a series of characters, rather than a string of bytes, the problem of the differing number of bytes used to represent Unicode is neatly sidestepped.

As mentioned earlier, the other problem with the different operating systems available today is the way they handle the end-of-line. The three main computer platforms all handle end-of-line differently—Unix uses the linefeed character, Windows uses a carriage return and a linefeed, while Macintosh uses just a carriage return. This difference can create problems when transferring files between platforms and definitely creates problems with byte-oriented authentication tools. The solution to this problem is to ignore it: this is explained next.

2.2 Dichotomous structure of markup files

The JITAM algorithm is also based on the premise that there are two readily identifiable types of information stored in a file that has been marked up with an SGML-like markup language. It is a requirement for all such markup languages that the embedded tags be identifiable for processing and parsing by a computer. Logically this entails that the content in the file must also be identifiable. In effect, this gives us two distinct streams of information stored in one file: the markup stream and the content stream. The requirement for authentication is to be able to detect meaningful changes in the content stream while ignoring changes in the markup stream.

3 Description of the Algorithm

The JITAM supports files using SGML-like markup languages. It is both platform and character-set independent in being able to detect changes in the textual content of a file. It treats the files as two intertwined streams of data, one containing the content and the other containing the markup.

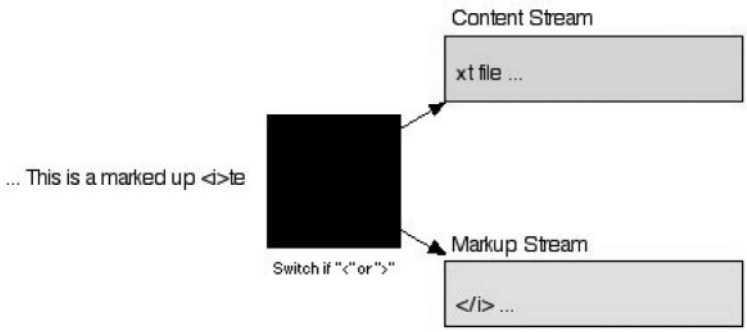

Fig. 1 The Finite State Machine

Computationally, the part of the algorithm that identifies the two different streams of data is in effect a Finite State Machine.[11] The machine accepts a single stream of characters (e.g. the contents of a file) as input. The output comprises of two streams of characters. These streams of characters are directed into one of the two containers with the choice of the container dependent on the nature of the characters in the input stream. This is shown in Fig. 1.

The result of this process is that all markup is put into one stream while everything else is put into the other. It is important to note that the stream of characters in the content stream container still has the same sequence as in the original file. This is the main requirement for the JITAM routine.[12] Having separated the markup from the content, the markup can now be ignored. Our sole concern here is that the characters of the content stream and their sequence have not been corrupted.

The text stream of a document is defined as a sequence of printable (or meaningful) words, together with any adjacent punctuation marks, separated by single spaces. This is in accordance with common practice for SGML applications[13] and gets around the end-of-line problem mentioned above, as ends-of-lines are literally ignored.[14] Programmatically, this is enforced in the algorithm by normalizing the white space in the separated content stream. Leading and trailing white-space characters are trimmed and runs of white-space characters between words are reduced to a single space character.[15]

Authentication of the content is done by calculating a checksum for the stream of content characters and comparing the calculated value against a previously calculated value that was arrived at using the same algorithm. If the two values do not match then there has been an alteration in the expected sequence of characters.

The algorithm's default *checksum* is calculated using the following mathematical function (known as a one-way hash function):

$$checksum = \sum_{i=1}^{l} i \times (ord(char_i) - ord(space) + 32)$$

where

'*i*' is the ordinal position of a character in the string.
'*l*' is the length of the string.
'*ord*(*chari*)' is the ordinal value of the ith character in the string.
'*ord*(*space*)' is the ordinal value of the space character in the character set.

This function steps through the content stream, character-by-character. The ordinal value of the character has the ordinal value of the space character in the character set subtracted from it to negate any problems with different character sets having different numerical values for the character. For example, the character 'f' has the ordinal value 102 in the ASCII character set and the ordinal value of the space character in the ASCII character set is 32 (the addition of the extra 32, which appears to effectively nullify the value of the space character is

explained shortly). The result of this equation is multiplied by the value '*i*', the character's position in the string, and added to the *checksum* variable. Therefore, the letter 'f' at the 10th position in a string contributes 1020 to the *checksum*, but an 'f' at the 12th position in a string contributes 1224. At the 100th position in the string, the same character would contribute 10,200 to the *checksum*. This multiplication step is a crucial part of the function as it is a way of recording a character's position—and as a result, the longer the string, the larger the *checksum*.

Lastly, the checksum function includes the addition of a constant offset value of 32. This offset value is applied for a number of reasons: the main one is that the space characters that are used to separate words contribute to the checksum for the string of characters. If this term was not added, a space character would contribute a value of zero to the checksum and this would not accurately represent the character string. The value of 32 was chosen so that the algorithm, which has only recently been made character-set independent, calculates the same checksums as a previous version that assumed the ISO-646 character set. The offset value means that a space character in the text equates to a numerical value of 32, no matter which character set is being used. This matches the numerical value for a space, in both the ASCII and ISO-646 character sets, and means that existing data using the previous algorithm still authenticate with the new function. A useful side-effect of the addition of this offset is that there is less likelihood that identical checksums will be calculated for short strings of text.

It should be noted that more than one string of characters can generate the same checksum. This is because there cannot be a one-to-one mapping between the numerical value calculated by the checksum function and all possible text streams that could be created. However, the chances of this happening *and* the string of characters making sense as words *and* fitting the context of the text become increasingly remote as the streams of characters being authenticated get longer.[16]

The checksum function used is based on one of the simplest and earliest authentication technologies developed for computers. This class of checksum function is called a Manipulation Detection Code (Christoffersson *et al.*, 1988) and is used for detecting accidental corruption of data rather than malicious modification. This sort of checksum function has been replaced by more elaborate functions for computer security applications because the likelihood of two different streams of text having the same checksum value is not considered small enough for those applications. However, such authentication technologies are concerned with the deliberate and malicious tampering of data, whereas JITAM is mainly concerned with the accidental modification of a file in a way that could also possibly be construed as a correct reading of the text. The chances of this happening are very remote indeed. A proper mathematical analysis of this likelihood would be very complex and beyond the scope of this introductory communication, but as an empirical example, an analysis of all the text elements from all states of Marcus Clarke's *His Natural Life* (HNL) as stored on the ASEC's JITM site was performed.

There are a total of 15,982 uniquely different text elements in the seven states of the text held on the site. In HNL, text elements are defined at the general level of paragraphs, which range in size from a single word to many hundreds of words. In all these text elements, there are a total of fourty-three cases where two different text elements evaluate to the same checksum using the hash function described previously. Of these, there is only one occurrence where the two different strings could be construed as a valid reading within the context of the text. Based on these numbers, the chance of clashing checksums is approximately 1 in 8,000. But the sample set is far too small for this statistic to be meaningful as a true indicator of the likelihood of an incorrect reading being accepted as a correct reading.

The two similar strings that did match checksums are, in fact, variant readings of the same part of the story from the English first edition and the serialized version in the *Australian Journal*:

> Checksum = 110872
> HNL-E1-3-23. TE-30: "Well bo's," said Gabbett, "what's to be done now?"

HNL-AJ-4-26. TE-66: ''Well, bo's,'' says Gabbett, ''wot's to be done now?''

This is exactly the type of misreading this technology is trying to avoid. Note that these are not strings with few words, as might be expected, and not just simple variants. There are actually three differences between the strings, which nevertheless make perfect sense in context. Fortunately, this problem can be solved at the implementation level without actually having to modify our basic algorithm or lose the character-set independent properties of our hash function.

By using the same hash function and calculating the checksum for a string in both the normal sequence and the reverse order, we can generate a much more secure authentication key for our text strings. For a clash to occur now, two strings would have to match with two different calculated values, which if we use the simplified probabilities from the HNL sample set mentioned previously, would be approximately 1 chance in 64 million (e.g. 8,000 squared). For the HNL sample set there were no examples of different strings that matched for checksums calculated both the forwards and backwards. This extension to the algorithm is still under consideration as to whether its use is warranted in our production JITM systems: further information concerning this development can be found on the JITAM website, cited later.

4 Implementation of the Algorithm

This JITAM algorithm already has a number of implementations. We have chosen the PERL scripting environment as the means of making it generally available. PERL is available on all current major computing platforms and the script is a fully commented, human-readable file. We are making this software freely available to the general academic public so that, within the limitations of the licence under which it is being released, it can be further developed for the good of all. The only platform-dependent aspect of this implementation is that it requires PERL version 5.8.1 or later to gain access to PERL's Unicode implementation and conversion routines.

The PERL script described in this article goes beyond authenticating the textual content of a file because simply being told that there has been a change to the file does not help find where the change has occurred. With the addition of an extra attribute to the elements used to define the structure of a file, this script can be used to not only check the authenticity of the content of the file at a more granular level, but also to check its overall integrity.

The Text Encoding Initiative DTD allows the inclusion of an 'id' attribute in all its elements. Similarly, other SGML-like markup languages, such as HTML, can incorporate user-specified attributes. Our script takes advantage of this feature. The 'id' attribute is used to identify uniquely an element and its contents within a file. This script is designed to authenticate only the content of elements whose start tags contain an id attribute.[17] This feature enables the creator of the file to authenticate transcribed material within these identified elements, but still make changes to content outside these elements. The unique identification of elements containing text to be authenticated also allows this script to report more accurately where accidental modifications have occurred.

Protection of files using this script is done first by preparing the file with id attributes as described above, and then generating a 'keys' file for the file using the script. This will produce a file that looks like the following:

```
HNL AE Prologue.tspt:26024905737
div1:HNL-AE-Prologue.TE-1:10197
div1:HNL-AE-Prologue.TE-2:3167
div1:HNL-AE-Prologue.TE-3:6583564
div1:HNL-AE-Prologue.TE-4:25816515
div1:HNL-AE-Prologue.TE-5:806960
div1:HNL-AE-Prologue.TE-6:13453767
div1:HNL-AE-Prologue.TE-7:1751143
div1:HNL-AE-Prologue.TE-8:536315
div1:HNL-AE-Prologue.TE-9:9078268
div1:HNL-AE-Prologue.TE-10:128013
div1:HNL-AE-Prologue.TE-11:8365242
div1:HNL-AE-Prologue.TE-12:186653
```

The first line contains the name of the target file and a JITAM checksum of the entire file. The following lines identify those elements within the file that have

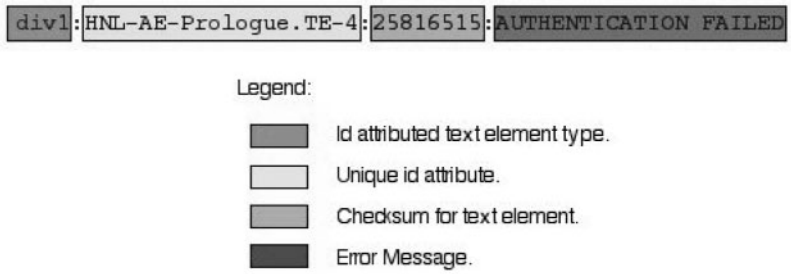

Fig. 2 Description of Keys File Entry

id attributes. Each line is broken up into a number of fields separated by colons. The first field identifies the type of the tag that has the id attribute. The second field gives the value of the id attribute for the element. The third field holds the calculated value of the checksum. Appended to each line after the third field will be any comments generated by the script when checking the file at a later time.

Having created the keys file, the target file may be modified (for instance, by adding markup) with any text editor. When the editing session is finished and the authenticity of the content needs to be checked, one simply runs the script again and looks at the keys file. If any corruption has occurred then a warning message will appear appended to the line of the relevant element as shown below:

HNL AE Prologue.tspt:26024905737:
AUTHENTICATION FAILED
div1:HNL-AE-Prologue.TE-1:10197
div1:HNL-AE-Prologue.TE-2:3167
div1:HNL-AE-Prologue.TE-3:6583564
div1:HNL-AE-Prologue.TE-4:25816515:
AUTHENTICATION FAILED
div1:HNL-AE-Prologue.TE-5:806960
div1:HNL-AE-Prologue.TE-6:13453767:
TEXT ELEMENT OUT OF ORDER:
AUTHENTICATION FAILED
div1:HNL-AE-Prologue.TE-7:1751143:
TEXT ELEMENT OUT OF ORDER:
AUTHENTICATION FAILED
div1:HNL-AE-Prologue.TE-8:536315
div1:HNL-AE-Prologue.TE-
9:9078268:AUTHENTICATION FAILED
div1:HNL-AE-Prologue.TE-10:128013:
TEXT ELEMENT NOT FOUND:
AUTHENTICATION FAILED
div1:HNL-AE-Prologue.TE-11:8365242:
TEXT ELEMENT NOT FOUND:
AUTHENTICATION FAILED
div1:HNL-AE-Prologue.TE-12:186653

In this example there are three types of failure all of which could be caused by mistakes while editing.[18] The first thing to notice is that, the first line shows that the authentication for the whole file has failed. This is to be expected as there are problems within the file. However, an authentication failure warning can appear in this line even though none of the other lines report failure: this indicates that a change in content of the file has occurred outside the uniquely identified elements that the algorithm checks.

The warning for the fourth of the identified elements indicates a textual corruption has occurred. This is the normal type of error the algorithm is meant to find and is explained in detail in Fig. 2.

The warning for the sixth and seventh text elements indicates that the elements have been identified as being in the file, but they are not in the right order. In this case they have been switched. The last error is catastrophic. A piece of content has been removed from the file starting somewhere in the ninth element. The tenth element can not be found at all and the integrity of the file is not re-established until the twelfth text element.

All the original data for the keys file remains in the file. If the problems with the target file are

corrected and the script run again, then the error messages will disappear. This can be done iteratively as problems are discovered and corrected.

5 Further Developments

The publication of this script gives anyone else who wants to work with this algorithm a place to start. The script is publicly available under an open source licensing model based on the BSD Open Source licence template. We invite users and developers to experiment with the script and modify it. The details of the licence and its restrictions are to be found in the script itself.

The latest publicly available version of the software is available on the following web page: http://www.unsw.adfa.edu.au/ASEC/JITAM/[19]

There are plans to incorporate the facility engendered by this algorithm into a GUI text editor using the Runtime Revolution development environment so that the same application will be usable on all platforms where the Runtime Revolution engine is available. This should make it available to users who are not happy with using a command-line interface.

6 Conclusions

It may be thought that the development of this authentication tool is at odds with the JITM system, one of whose guiding principles is simplifying the process of marking up, by abstracting markup away from the transcription files. It is our belief that the benefit of the stand-off markup mechanism with its greater versatility will prove itself in the long-term and become more accepted as people discover the inherent limitations of the current paradigm.

However, the inertia of the current paradigm, and the software tools and businesses that have been developed to support it, cannot be easily redirected, so it is relevant to note that the steps required to use the JITAM authentication algorithm also constitute preparatory steps for the conversion of such files to a JITM system. Adopters of this tool gain this additional benefit with its use.

The development of this algorithm has also benefited JITM. Originally, the strength of the JITM system was that it could authenticate its base transcription files prior to embedding its stand-off markup, thereby testing the integrity of the system. The JITM system now also authenticates the content of its generated Perspectives after they have been created using the JITAM algorithm, thereby giving a further level of trust to users of JITM.

Finally, the JITAM algorithm, when further developed, will potentially allow the veracity of all instances of a work to be checked no matter on what platform they have been instantiated, or how heavily they have been marked up. Thus a logical equivalence mechanism is being posited, whereby multiple versions of a document might exist, each containing different markup or even using different SGML-like markup languages,[20] but all of which can be automatically verified as being generated from, and remaining faithful to, the same base text. In the digital world, where ease of manipulation and dissemination are so frighteningly easy, this idea and perhaps this algorithm will provide a much needed measure of trust for digital resources.

References

Barwell, G., Tiffin, C., Berrie, P., and Eggert, P. (2003). The Authenticated Electronic Editions Project. In *Computing Arts: Digital Resources for Research in the Humanities. Papers from a Conference held at The University of Sydney, September, 2001*. Sydney.

Christoffersson, P. *et al.* (1988). *Crypto Users' Handbook: a Guide for Implementors of Cryptographic Protection in Computer Systems*, Viiveke Fak, (ed.), Elsevier Science Publishers B. V.

Rivest, R. (1992). *The MD5 Message-Digest Algorithm*. http://www.faqs.org/rfcs/rfc1321.html (accessed 6 June 2005).

Notes

1 Funding for these projects was provided by grants from the Australian Research Council.

2 Markup is limited to the definition of text elements with a unique id attribute to assist in text manipulation and identification and the encoding of non-ISO-646 characters as entity references.

3 Having texts and data at multiple sites, and still being able to ensure their integrity, also provides insurance in case of catastrophe at one of the sites.

4 The JITAM algorithm was developed in response to a question regarding the possibility of a similar tool to provide continuing authentication for electronic texts being developed using the current paradigm of all-in-one markup. The original request was made by Professor Hugh Craig of the University of Newcastle, NSW, during the Computing Arts 2001 conference in Sydney. The initial response was negative, but that started us thinking on the idea.

5 UTF stands for unicode transaction format. UTF-8 is a variable length format where the characters found in the standard ISO-646 and ASCII character sets are represented by 1 byte. Characters outside this set of characters are represented by 2 or more bytes. UTF-8 is the default character set for XML. It is important to note that UTF-8, UTF-16 and UTF-32 are different: UTF-16, always uses two bytes and UTF-32 uses four bytes to represent all characters. Only UTF-8 uses a variable number of bytes to represent Unicode characters.

6 MD5 stands for Message Digest 5 and is commonly used for checking the data integrity of static files. The MD5 algorithm was developed by Professor R.L. Rivest at MIT (Rivest, 1992) and has been widely accepted. Most computing platforms have an implementation of this algorithm.

7 A checksum is a numerical value generated from a string of bytes in a computer. Checksums can be used to represent the original string of bytes from which they were created. This idea is the basis of many authentication technologies.

8 A bit-oriented authentication code uses the fact that each character has a numerical equivalent. A space character has the numerical value of 32 in ASCII. The 'A' character has the numerical value of 65 while the numerical value of 'a' is 97.

9 Big-endian and little-endian storage order is an arbitrary decision made by hardware vendors. If files are properly converted in transferring between platforms then there is no problem. However, if the conversion is not done, the file is more or less unreadable by the other platform. Most mainframe computers are big-endian while most Intel-based PCs are little-endian.

10 The common alphabet (i.e. a–z) in its normal ordering is a subset of the ISO-646 character set. ISO-646 also contains other characters including the uniquely-defined, uppercase versions of the alphabet.

11 A Finite State Machine (or Finite State Automaton) is an abstract model used in computational studies. It represents a machine with finite resources that changes its operation (or state) based on its input data. In effect, all computers are Finite State Machines, but this type of computational model is commonly used in language parsers and compilers, where the interpretation of the data is based on the state the machine is in at the time the data is received. This allows a file to contain both data and program instructions written in the one language (e.g. the code for a program).

12 The diagram displays the text stream in readable order. The real process would be occurring in the opposite order as characters are read and processed consecutively from the beginning of the file.

13 HTML, the principal SGML application in use today, follows this maxim and browsers are meant to ignore non-printing characters in a file. The corollary of this is that typographic white space has to be coded (e.g. '
' for end-of-line and ' ' for non-breaking space).

14 And relevant end-of-line formatting need to be represented by a suitable tag.

15 White-space characters are defined as those that have no visible glyph. These include spaces, tab characters, and end-of-line characters.

16 This proviso applies to all numerically based authentication methods to a greater or lesser extent and is a limitation based on the nature of the problem, the limitations of computers, and the largest numbers they can store in memory.

17 Currently the script does not support tags with the id attribute that are enclosed within id attributed tags. Future versions will address this problem.

18 This script is also useful for detecting changes to a file that occur through the use of faulty automated systems and errors in file transfer.

19 Please direct any comments or suggestions regarding this algorithm or its software to the following email address: acadedns@adfa.edu.au

20 For example, JITAM could be used to prove that identical base transcriptions were used for both a scholarly edition of a text, complete with TEI markup, and an eBook reader version.