Hash-Tree Anti-Tampering Schemes

Ben Moss and Helen Ashman

Abstract--**Procedures that provide detection, location and correction of tampering in documents are known as anti-tampering schemes. In this paper we describe how to construct an anti-tampering scheme using a pre-computed tree of hashes. The main problems of constructing such a scheme are its computational feasibility and its candidate reduction process. We show how to solve both problems by the use of secondary hashing over a tree structure. Finally, we give brief comments on our ongoing work in this area.**

Index Terms--**Computer Security, Hashing, Tamper Correction, Trees.**

INTRODUCTION

As the volume of information existing in digital form increases, data tampering becomes a growing problem. Current tamper protection schemes employ cryptographic hash functions, such as SHA-1 [1], to protect data integrity. Such functions create a digest of the data, which can be used to determine the data's integrity at some later point in time. Instances of tampering detected through this method result in the data being discarded, and (in the case of transmitted data) a request for retransmission is made.

Using cryptographic hash functions to protect against tampering is very effective, but there are some disadvantages. For example, assume that an e-mail has its subject field modified. The current method of tamper protection would identify that some form of tampering had taken place. However, the recipient of the e-mail cannot determine which parts of the information have been tampered with and which parts are original: the whole e-mail is rendered suspect. The recipient may have only been concerned with the e-mail's content, and therefore request for retransmission is actually unnecessary. In such a case, the ability to locate and correct tampering would make retransmission unnecessary.

Another problem with the current method involves the urgency or time-scale validity of information. If tampering has occurred, then it may not be feasible to request retransmission of information; and there are no guarantees that further transmissions will be tamper-free. It would often be more practical to reconstruct the original information from the data that has already been received, similar to the situations where error-correcting codes are used (see related work). However, this work concerns the recovery of data subject to *deliberate tampering*, and has no links with error-correcting codes.

This work was supported in part by British Telecom.

B. Moss is with the School of Computer Science and IT, University of Nottingham, Nottingham, UK (e-mail: bxm@cs.nott.ac.uk).

H. L. Ashman is with the School of Computer Science and IT, University of Nottingham, Nottingham, UK (e-mail: hla@cs.nott.ac.uk).

ICITA2002 ISBN: 1-86467-114-9

We first discuss anti-tampering schemes in general, describing how they provide an alternative to normal cryptographic hashing, but with added advantages. We then discuss how arbitrary documents can be represented with tree structures, before describing hash trees. The related work is then investigated, and we conclude with a summary and report on the further work currently in progress.

ANTI-TAMPERING SCHEMES

Tampering, like other areas of computer security, can be divided into prevention, detection, location and correction. Many schemes claim to prevent tampering, but in fact only detect the presence of tampering.

This work concerns so-called *anti-tampering schemes* that not only detect tampering but attempt to locate and correct it too. Since we are interested in schemes robust enough for data transmitted over the Internet, we do not pursue tamper prevention, as this would involve protecting data by some physical means. Therefore, we assume that tampering has already occurred and attempt to solve the following problems:

1. Detection - identifying that the document has been modified;
2. Location - locating the point(s) at which the document has been modified;
3. Correction - restoring the modifications in the document to their original state.

Assuming correction is done sensibly, so that only modified parts of the document are corrected, then the three problems are hierarchical: Locating where tampering has taken place also implies that tampering has been detected; and correcting the tampering also implies that tampering has been located.

The idea for an anti-tampering scheme was first postulated by Ashman [2]. The fundamental idea is to check a document's integrity at different levels, rather than the integrity of the document as a whole. Tamper detection is achieved in the same way as current systems, by hashing the entire document using a normal cryptographic hash. Subsequent tamper location is achieved by multiple hashing applied to subsections of the document. Tamper correction is achieved by using small pre-image hashing of the subsections, which allows a brute-force search through all possible subsections to determine the original.

Anti-tampering schemes are suitable for data at risk from tampering, whether stored or transmitted. In particular, they have application to data transmitted asynchronously with a time-scale validity, where request for retransmission in the presence of tampering would exceed the time-scale, or would be inconvenient.

For example, suppose that a message in a financial transaction protocol contains an account number field. On route to its destination, the message is tampered with and the account number is changed. An anti-tampering scheme could:

1. Detect that the message had been tampered with;
2. Locate that the tampering had taken place in the account number field, at the same time verifying that other fields in the message were unchanged;
3. Correct the message by restoring the account number field to its original state.

As with normal cryptographic tamper detection, in anti-tampering schemes it is necessary to store and/or transmit the file and the hashing data separately. If this were not the case, an attacker could modify the data and then compute the hash tree for the modified data to bypass the anti-tampering scheme.

Document Trees and Hash Trees

Dividing data into subsections recursively gives rise to a hierarchical tree-structure of increasingly small subsections. Some data is inherently tree-structured. For example, if the document was a book, then the root node's data is the whole book. The book can be split into chapters, so the data of the root's child nodes is each chapter. Each chapter can be split into paragraphs, so the data of a chapter's child nodes is each paragraph. This recursive decomposition process can continue until each word is split into characters. It obviously ignores elements such as punctuation, but the concept of recursive data decomposition is clear.

Not all data has a natural tree-like structure, but this does not impact on the efficacy of anti-tampering schemes, as we are able to superimpose tree structures onto data. This can be done in the same way that quad-trees are often superimposed onto images and other two-dimensional data [3]. However, we do not restrict anti-tampering schemes to quad-trees alone. Data for such schemes can be divided into binary trees, *k*-trees (where *k* is a constant branching factor) or even *x*-trees for variable branching throughout the tree (such as the book structure in the previous example). The application of these structures simply provides a method for data decomposition within anti-tampering schemes; it does not interfere with the original data in any way.

Given that any data has a tree-like structure (whether natural or superimposed), then any document has a *document tree*. The root node's data of the document tree is the whole document's data. This is then broken into distinct elements according to the document's tree structure. The data of these distinct elements is then the data of each child node. This is repeated recursively to an appropriate level dependent upon the type of document. Each level in the document tree contains all the data needed to create the entire document, but it is split into a greater number of elements than the parent level.

The idea suggested by Ashman is to pre-compute a *hash tree* (different from hash trees used in [4]), which has the same structure as the document tree, but differs in the data stored at each node [2]. The data stored in the nodes of a hash tree are hashes of the data stored in the corresponding nodes of the document tree. Therefore the hash tree is much smaller than the document tree (although not necessarily smaller than the document). The computation of a single cryptographic hash (such as SHA-1) stored at the root of the hash tree is enough to determine the legitimacy of a document. This is referred to as the *primary hash*. *Secondary hashes* (which have a smaller image and consequently are less cryptographically robust) can be used below the root node, since the primary hash will always ultimately determine fake documents. Having pre-computed a document's hash-tree it can be used at a later stage to detect, locate and correct any tampering.

Hash-Tree Anti-Tampering Schemes

Tampering of a received document is detected by comparing its computed hash with the received hash tree's primary hash. If the hashes are identical, then the document has not been modified at all, otherwise tampering has occurred. This is identical to current tamper detection methods that use cryptographic hashing. However, anti-tampering schemes also provide the ability to locate and correct tampering.

To locate any tampering, the next level of the received document's document tree is calculated and the respective hashes are computed; these can then be compared to the original hashes in the received hash tree to detect which sub-sections of the document have been modified. This process is repeated down the document tree and its respective hash tree to locate exactly which parts of the document have been changed, until the lowest level of the hash tree has been reached.

In order to correct any tampering we have located, we must guess the original data at that point in the tree by brute force. By computing the hash of each guess and comparing it with the corresponding hash at that point in the hash tree, we can determine the guess to be a *candidate* if the hashes match (otherwise we discard the guess). As hashing at this level is secondary hashing, it is more probable that multiple candidates occur. After finding all possible candidates, we construct candidates for the parent data (incorporating every combination of remaining child candidates) and test each one by computing its hash and comparing this to the parent hash in the hash tree. By repeating this process recursively upward through the tree as many times as necessary we can reduce the set of candidates to one and thus determine the original data. Note again that, ultimately the primary hash will determine a correctly reconstructed document.

The use of secondary hashes keeps the size of the hash-tree file relatively small, whilst increasing the number of guesses which qualify as candidates during tamper correction. As such, a tradeoff between hash-tree file size and reconstruction time exists.

In general, it is considered computationally intractable to guess the input that corresponds to a given cryptographic hash by brute-force. However, in the case of hash-tree anti-tampering schemes the data is divided into small enough leaves (with known upper bounds) at the lowest level of the document tree, that a brute-force search of every possibility becomes feasible.

Part of our ongoing work is to find the optimal balance between minimizing the hash-tree file size whilst allowing reconstruction of the original data in reasonable time (assuming that tampering has occurred).

There is one situation where the use of anti-tampering schemes can take advantage of both small transmissions and computationally lightweight file reconstruction. In some applications, it is not necessary to transmit the entire hash tree, but only those parts directly relevant to reconstruction. So if tampering is detected in a file, the immediate child hashes are requested, checked against the received file, and then only the necessary child hashes of any tampered subsection would subsequently be requested. Such situations have the additional overhead of multiple transmissions, but the hash-tree file size becomes less important as only relevant parts of it are transmitted.

Related Work

There is very little work that looks into the correction of documents that have been deliberately tampered with. This may be because tamper-detecting hashes need to be cryptographically robust and, as such, are relatively large in size, whilst at the same time they are non-deterministic, and so there will be many other documents with the same hash. Also, cryptographic hashes do not identify where the tampering occurred, only the fact that a change in the document has occurred since the hash was computed.

A. Error-Correcting Codes

Detecting where changes have occurred and reconstructing the original data requires redundant information, a concept familiar to practitioners of error-correction. The positioning of the redundant data throughout the original data makes it possible to locate the change, while the content of the redundant data makes it possible to deterministically reconstruct the original data.

On the other hand, error-correcting codes are not robust enough to overcome tampering, but are optimized for maximum correcting capability with minimum overhead. It would be a simple matter to change the original content of the data and to change the error-correcting data so that the transmission appeared to be legitimate. Thus error-correcting codes, which have been designed to detect and correct accidental transmission errors, are not proof against deliberate tampering.

In contrast, anti-tampering schemes do not embed the redundant data, but create a totally separate file whose structure mimics that of the true data. As with normal cryptographic hashing, it is necessary to separate the hash data from the file, so that an attacker cannot change both the file and the hash data. Applications such as PGP do this by digitally signing the hash data, but an alternative is to separately transmit the hash data, for example by some form of network diversity [5].

B. Message Recovery Schemes

Other related work is the idea of the message recovery schemes, proposed by Nyberg and Rueppel [6]. However this does not apply to general tamper-proofing of arbitrary files, but provides a means for embedding a message in a digital signature or similar cryptographically robust construct. However, in this scheme it is not possible to actually reconstruct a message after tampering, rather it is “recovered” from the cryptographic data. If tampering has occurred, it would be necessary to request a retransmission.

Summary and Work in Progress

This paper outlines the notion of anti-tampering schemes, which allow the reconstruction of a tampered file to its original state from a combination of the tampered file and a pre-computed hash tree.

A project currently underway is the use of non-recursive document segmentation, so that the divide-and-conquer approach operates on overlapping document sections rather than nested sections. This project uses *n*-grams as document sections, with an *n*-gram being a sequence of *n* bits from a file. Simple hashes are calculated over each *n*-gram. The overlapping of *n*-grams means that each individual *n*-gram contains a unique sequence of bits from the file. It is possible to detect which bit has been changed by looking at the hashes of all *n*-grams containing that bit. This iterative variant allows one to apply anti-tampering to stream data.

Additionally, we are currently constructing a generic model for hash-tree anti-tampering schemes with the aim to optimize storage and operation time. We encourage other work in the area of anti-tampering schemes, particularly in the design of secondary hash functions for use in such schemes.

References

[1] U.S. National Institute of Standards and Technology, “Secure Hash Standard,” NIST FIPS PUB 180-1, April 1995.

[2] H. L. Ashman, “Hashes DO Grow on Trees - Document Integrity at Every Level,” Proceedings of Ausweb 2000 (http://ausweb.scu.edu.au/aw2k/papers/ashman2/index.html), Southern Cross University, June 2000.

[3] Hanan Samet, “The Quadtree and Related Hierarchical Data Structures,” Computing Surveys, Vol. 16, No. 2, June 1984, pp. 187-257.

[4] R. Merkle, “Protocols for Public Key Cryptosystems,” Proceedings of IEEE Symposium on Research in Security and Privacy, Oakland, April 1980.

[5] H. Ashman, and M. Gilbert, “And now for something completely different: looking ahead to new encryption and secrecy protocols,” Proceedings of Communications Design Conference, October 2001.

[6] K. Nyberg, and R. Rueppel, “A New Signature Scheme Based on the DSA Giving Message Recovery,” First ACM Conference on Computer and Communications Security, ACM Press, 1993.