The Steganographic File System

Ross Anderson[1], Roger Needham[2], Adi Shamir[3]

[1] Cambridge University; rja14@cl.cam.ac.uk
[2] Microsoft Research Ltd; needham@microsoft.com
[3] Weizmann Institute; shamir@wisdom.weizmann.ac.il

Abstract. Users of some systems are at risk of being compelled to disclose their keys or other private data, and this risk could be mitigated if access control mechanisms supported an element of plausible deniability. However, existing plausible deniability mechanisms, such as the one-time pad, are of rather limited scope.

In this paper, we present the steganographic file system. This is a storage mechanism designed to give the user a very high level of protection against being compelled to disclose its contents. It will deliver a file to any user who knows its name and password; but an attacker who does not possess this information and cannot guess it, can gain no information about whether the file is present, even given complete access to all the hardware and software. We provide two independent constructions, which make slightly different assumptions.

1 Introduction

Much work has been done on devising mechanisms, such as digital signatures, that can be used to provide non-repudiation; there has been much less work on the complementary property, namely plausible deniability. Yet there are many applications in which plausible deniability could be valuable:

- soldiers and intelligence agents may be captured and tortured into revealing cryptographic keys and other secret data;
- when conducting delicate negotiations, such as between a company and a trade union, informal offers may be made which will be denied in the event of later litigation. However, the other side might obtain court orders for access to documents;
- police power may be abused. An individual may be arrested on 'suspicion' of a crime, found with an encrypted hard disk partition, and told that if he does not supply the password, this will be taken as evidence of guilt. But the encrypted files might well contain confidential business information sought by people who have bribed the police;
- private individuals have been tortured by robbers into revealing information such as the secret codes for their bank cards and the location of safes [12].

There are few mechanisms available at present to provide protection against these threats. Crypto keys may be kept in tamper resistant devices, but tamper resistance is relative, especially against a capable motivated opponent [4]; incriminating documents can be shredded, but the word processor files used to create them may persist for months in backup tapes (as Oliver North found to his cost); and while the one-time pad may provide an element of plausible deniability in communications, it is difficult to use in data storage because the pad must also be stored somewhere (in the next section, we will discuss one way to overcome this difficulty). There are some deniable encryption schemes in the literature (e.g. [5, 6, 11]) but these protocols can still generally protect only short messages, and are not applicable to storage.

One possible defence against compulsion is dual control. Bank managers understand that the real purpose of having a dual combination lock on the vault is not to prevent them taking cash (they have many ways to do that!) but to stop their families being taken hostage. The manager's inability to open the lock on his own removes the temptation for criminals to try to force him.

Dual control may be inappropriate, such as for an attorney in single handed practice. It may also be inadequate, for example where crypto keys are shared between two soldiers in a signals unit who might be captured at the same time, or where someone has abused the legal process to obtain an injunction compelling an innocent party to disclose information against his interests. In all such cases, it may be sufficient if the victim can convincingly simulate an inability to perform the act required by the opponent.

These considerations motivate us to design a file system with the following property. A user may provide it with the name of an object, such as a file or directory, together with a password; and if these are correct for an object in the system, access to it will be provided. However, an attacker who does not have the matching object name and password, and lacks the computational power to guess it, can get no information about whether the named object even exists.

The concept of operations is that the user of such a file system could, if placed under compulsion, reveal (say) the three passwords used to protect the directories with his email archive, his tax records and his love letters, but keep quiet about the directory containing his trade secrets. The opponent would have no means of proving that such a directory exists.

We do not assume any properties of tamper resistance, whether of hardware or software. We assume that the opponents who may place the user under compulsion are competent; that they can examine the system at the level of bits and gates, and understand all its hardware and software completely. The only difference between them and the user is that the user knows one or more passwords (or more accurately, strong passphrases).

The first of our constructions assumes only that there are limits to the amount of knowledge of the plaintext that the opponent possesses, and to the number of computations that he can perform. In the second, we will assume the existence of a block cipher that the opponent cannot break.

2 A Simple Construction

Our first construction makes no assumptions about the existence of 'good' ciphers. Its protection goal is that if the opponent has no knowledge of the filename and password, then he can get no information about whether such a file is present in the system unless he already knows some of the file contents or tries all possible passwords. For the sake of simplicity in exposition, we will assume a password P of k bits. We might imagine that the filename plus a strong passphrase has somehow been compressed to a P of k bits in length; but we stipulate nothing about P other than that the opponent must not be able to guess it.

The idea is to have a number of cover files in the system, which start out as random, and then embed the user's files as the exclusive or of a subset of the cover files. This subset is chosen by the password.

To give a concrete example, suppose that all files are of the same length and that there are k cover files in the system to begin with, say $C_0, \ldots, C_{k-1}$. Let the first user file be F and its password be P. We then select those C_j for which the jth bit of P is one and combine them using bitwise exclusive or; the result is in turn XOR'ed with the user supplied file F and the result of this operation is finally exclusive or'ed with one of the C_j. The result is that the user's file F is now the exclusive or of the subset of the C_j selected by the nonzero bits of P. Symbolically,

$$F = \bigoplus_{P_j=1} C_j \tag{1}$$

A user can embed a number of files as combinations of the cover files. Adding subsequent files entails solving sets of linear equations to decide which combinations of the C_j to alter.

An important property of this system is that if we have a linear access hierarchy — that is, a user storing a file at a given security level knows the passwords of all the files stored at lower levels — then files can be added in a natural way without disturbing already hidden files. Assuming that the k cover files $C_0, \ldots, C_{k-1}$ are used to hide a smaller number m of user files $F_0, \ldots, F_{m-1}$, and that each file is n bits long, let $\mathbf{C}$ be the k x n matrix over GF(2) whose i-th row is the current contents of cover file C_i. Initially this matrix is filled with random bits. As we store or update an actual user file F_j, we modify the k cover files in a way which is guaranteed to change F_j without affecting any of the other user files; this can be done even by someone who has access only to files up to a certain security level, and has no information about the contents (or extraction method) of the "higher" files.

The basic idea is to use an orthonormal k x k binary matrix $\mathbf{K}$ as the extraction key. If K_j is the j-th row of $\mathbf{K}$, then F_j is defined as the vector-matrix product K_j x $\mathbf{C}$ (mod 2), which XORs the rows of $\mathbf{C}$ indicated by the 1's along the j-th row in $\mathbf{K}$. The orthonormality condition implies that K_j x K_i^t (row times column product, giving a scalar) is 1 if $i = j$, and 0 otherwise.

Suppose we want to modify only the i-th user file F_i by XORing it with the binary difference file D of length n. We change the cover file system represented by the matrix $\mathbf{C}$ by XORing to it the matrix K_i^t x D (^{t} denotes transpose, so this is a column vector multiplied by a row vector, giving a k x n matrix). Consider now for any j the new value of the user file F_j, computed by K_j x ($\mathbf{C}$ XOR K_i^t x D) = (K_j x $\mathbf{C}$) XOR (K_j x K_i^t) x D. The second part is zero for all $i \neq j$, and D for $i = j$, which gives the desired effect on the secret files.

What is left is to show how a user can be given only his part of the key matrix $\mathbf{K}$, without revealing other parts or asking him to memorize lots of bits. We can use the standard trick of mapping a random initial password p_0 by iterating a one way function h (or even better, a one-way permutation) via $p_{i+1} = h(p_i)$. If we now give some user the value of p_i as his password, then he can compute all the later p's but not the earlier p's. We now map each p_i into a random binary vector with an odd number of 1's (so that its dot product with itself is 1). Finally, we use the Gram-Schmidt method to orthonormalise all the vectors from i onwards by subtracting from the candidate K_i all its components along later K_j which the user knows by the chaining property of the p_j's. In other words, a user who knows p_i can compute all the orthonormal key vectors K_j for $j \geq i$, but not any earlier ones.

This gives us a linear hierarchy of access rights, where each user has access to all the files from some index onwards. We cannot extend it all the way to k secret files, since when $K_2, K_3, \ldots, K_k$ are known binary vectors and K_1 is known to complement them to an orthonormal basis, it is uniquely defined. But if we ensure that the number m of secret files is a number randomly chosen in $1, \ldots, k/2$, the orthonormalization process is very likely to succeed, and it will be exponentially hard in k to find one more key vector given all the later key vectors.

To extend this 'multilevel secure' file system to provide the plausible deniability which we seek, the user must have a number of passwords p_i rather than just one (or two) of them. The p_i can therefore be generated at random rather than by repeated hashing, and users can manage them in any of the standard ways: a number of the p_i could be stored on disc, encrypted under user passphrases, together with a number of nulls (objects that look like encrypted passwords but are not), so that the opponent cannot tell how many genuine ones there are. Alternatively, mnemonics can be used to map the p_i to passphrases, in which case there would be no need to keep passwords encrypted on the disk. The rest of the linear algebra goes over as before.

Linear algebra also gives us a known-message attack: if the size of the password is k and the opponent knows more than k bits of plaintext, then after obtaining all the random files from the computer he can write k linear equations in the k unknown bits of the key. This is why we add the explicit assumption that the opponent knows nothing about the plaintext. If this assumption holds, then the scheme appears vulnerable only to an opponent who can guess the password.

Requiring that the opponent be totally ignorant of the file contents may be simple where these contents consist of random data such as crypto keys, but could often be onerous. For example, a tax inspector searching for early drafts of a set of accounts might know the location and value of the tax reference number in the files in question, and this could be enough to break the system for (say) $k = 100$. There is a simple practical solution: to disallow knowledge of the plaintext by preencrypting it with a key derived from the password. We will discuss below how an alternative construction is available, given the assumption that strong ciphers exist.

Meanwhile, we have shown how a simple steganographic file system can be constructed; it enables a user under compulsion to reveal (say) seventeen passwords that retrieve files on his tax affairs, love affairs, and so on, but hold back the eighteenth password which would have revealed the design for the new security system that the opponent was interested in. On the assumptions above, and absent implementation defects, there is no way that the opponent can tell that a valid eighteenth combination of filename and passphrase exists at all in the system.

In a practical system, we may wish to provide for files of different lengths, and we will also probably want the granularity of protection to correspond to directories rather than individual files. This can be accommodated in the above system by packing each directory into one of the files F_j whose length might be fixed at (say) 10MB; 100 such directories easily fit on a typical modern PC with 3.5 GB of hard disk, leaving plenty of room for the operating system and non-secret software and data. We could then end up with a hierarchy of several dozen directories for 'tax', 'love letters', 'company secrets' and so on, will only have to remember one passphrase for each level at which disclosure is to be made. Disc space is nowadays so cheap that having to allocate large fixed size directories is not a real problem for most applications; however there are two problems, one major and one minor.

The minor problem is that it might not always be convenient to have a strict linear access hierarchy, and the major problem is that there would be a significant performance penalty: reading or writing a file would involve reading or writing about 50 times and if this involved processing the entire directory contents it would entail about 500MB read or written on average. It need not however, since the modification or access can be done on arbitrarily narrow "slices" of the k x n matrix $\mathbf{C}$. For example, if the modification vector D is nonzero in a single bit (say, the q-th), then the product k_i^t x D is nonzero only in its q-th column, and thus both the computation and the data access are unnecessary except at these k bytes.

A further reduction in the overhead of reading and writing can be obtained by doing arithmetic over a field other than GF(2). This makes it possible to separate the security parameter from the number of levels in the access hierarchy.

The reason we have 100 cover files is to have a complexity of 2^{100} in guessing the linear combination. This also gives us 100 levels in the hierarchy, which is likely to be too large. Instead, we can use 16 cover files which are added with coefficients from, say, GF(2^8) or GF(251) (the largest single byte prime). Guessing a vector of 16 such coefficients has probability about 2^{-128}, and we have about 8 levels in the access hierarchy. In terms of efficiency, to compute one byte of real file we need only 16 bytes (one from each cover file). This is probably a good choice of parameters. However, we still have a 16-fold penalty on reading and writing, and are constrained to a linear access hierarchy. This motivates us to ask if there is another way of implementing a steganographic file system.

3 An Alternative Construction

One might ask: why not just fill the whole hard disk with random bits, and then write each file block at an absolute disk address given by some pseudorandom process, such as by encrypting the block's location in the file using as a key a one-way hash of the filename and the directory password? The block could also be encrypted using a key derived using a similar process, and so — on the assumption that we have a block cipher which the opponent cannot distinguish from a random permutation — the presence or absence of a block at any location should not be distinguishable.

The problem with such a naïve approach is that, thanks to the birthday theorem in probability theory, we would start to get 'collisions' (where we overwrote already written data) once we had written a little more than $\sqrt{N}$ blocks, where there are N blocks in the disk. Given that a modern hard disk might have a million blocks, it would be 'full' after somewhat over a thousand of them had been written — a very wasteful use of resources.

However, there is a solution: write the block at more than one location. If, for example, we write each block in two pseudorandomly chosen locations, then both of these blocks will be overwritten by a third when the number of blocks is approximately that required to produce triples under resampling, namely $O(N^{2/3})$. It is straightforward to add redundancy to each block before encrypting it, so that the system can identify when a block has been overwritten and look for it in the other location instead.

Continuing in this way, we might hope that if we write to ten locations, then before all locations containing a block are accidentally overwritten, we could write $O(N^{9/10})$ blocks, and so on. However, the situation is somewhat more complicated.

Suppose that that K different blocks are written and that we write each block m times, giving a total of mK write operations (including overlaps). Consider the probability that an individual block will be overwritten by subsequent writes. If there are M of these, then the total number of distinct blocks that are written n_i times out of the total of N blocks is [1]:

$$\pi(n_1, n_2, \ldots, n_M) = 1/N^M \binom{N}{n_1, n_2, \ldots, n_M} \frac{M!}{\prod_{i=1}^{M} (i!)^{n_i}} \tag{2}$$

which can be approximated for computational purposes as:

$$\pi(n_1, n_2, \ldots, n_M) \simeq 1/(N^m e^{M^2/2N}) \frac{M^{M-n_1}}{\prod_{i=1}^{M} (i!)^{n_i} n_i!} \tag{3}$$

so the total number of distinct blocks that are overwritten is:

$$\Phi(M, N) = \sum_{1}^{M} \pi(n_1, n_2, \ldots, n_M) \tag{4}$$

where the sum is over all possible combinations of the n_j. The probability that one of the k versions of block number j survives this overwriting process is given by $1 - [\Phi((j-1)m, N)]^k$, and indeed the probability p that any one of the K files gets overwritten is:

$$p = \sum_{j=1}^{K} [\Phi((j-1)m, N)]^k \tag{5}$$

No analytic solutions are known for equations of type (2)–(5), and so we do not expect to find a closed form solution for the number m of replications that will maximise the 'load factor' of our system — this is K/N, the ratio of the total number of different blocks stored to the total disc space allocated to the system. However, a numerical model suggests the following results. If we consider the disk to be full the first time a block is corrupted (in that all copies of it are overwritten), then the load factor is rather low, with about 7% load achieved for $m = 4$ over a range of values for the number of blocks N from 20,000 to 1,000,000. This figure improves dramatically as soon as we allow even a small number of blocks to become corrupted; and if we allow 10% of blocks to be corrupted, we can obtain a raw load factor of 20% — giving an effective load factor of perhaps 15% once forward error correction is applied. These figures are somewhat tentative and we expect that they could be improved given an implementation. There are however some remarks that can be made in advance of that.

Firstly, this construction largely overcomes the performance problem of our previous construction. Files are still read and written multiple times, but the multiple is now about 5 rather than the 16–50 range of the first construction. Furthermore, we have a critical trick, in that when operating in a given directory (i.e., using a single user password, and operating at a single security level), we keep a Larson table which tells us which block to look at first [10].

Larson's system was designed to allow any record in a database to be retrieved with only one disk access. The basic idea is that a record is written at one of

m locations on disc, which are specified pseudorandomly as in our construction, and a table is kept in memory telling the user at which location to look. In more detail, a record with key K is hashed to provide a 'probe sequence' of m values $h_1(K) \ldots h_m(K)$, plus a 'signature sequence' of m values $s_1(K) \ldots s_m(K)$; the latter are each of k bits in length. A table of 2^k values is kept in memory. A record with the key K is stored at the address $h_1(K)$, unless it is already full, in which case an attempt is made to store it at $h_2(K)$, and so on. The signature sequence is then used to encode the record's location in the table in memory. If all the addresses $h_1(K) \ldots h_m(K)$ are occupied, then the system declares 'disk full'. Experiments by Larson and Kajla showed that with values of m in the range 10–25, the disk would not be full until 80–90% of its blocks were occupied.

So a Larson table at a given security level (i.e., directory) can be built whenever the directory is opened and which points to locations which contain a true copy of each block; these can simply be the first uncorrupted copy of the block that can be found. This block can also be written first should the application write the block back. This table building can be carried out as a background process, so there need only be a performance penalty for those files which the user opens immediately on changing to a new directory. The table builder can also alarm when it first finds that a block has been completely corrupted (the disk is full) or that the limits of the error correction code are being reached (the disk is nearly full).

The performance of file systems based on Larson tables is relatively well understood. Reading a file that is stored at hashed absolute locations indexed by a Larson table is often faster than with a conventional file system, as the table can be held in memory and only one disk access is needed to read any block. On the other hand, writing is slightly slower. In the context of the steganographic file system, there is extra overhead when we close a file; blocks that have been written need to be replicated. We expect that the replication can largely be done in background, unless the user closes a file and changes immediately to a directory with a lower classification.

Secondly, as in our first construction, the assumption of a linear access hierarchy enables us to improve things substantially. In the present construction, the improvement is that we can greatly increase the load factor. Consider the lowest level in the hierarchy; the password for this directory will be known to all higher level processes, and so will always be available whenever a file is being written. Thus higher level processes can avoid the blocks of the lowest level files, and in consequence the lowest level directory can be implemented using Larson tables directly with no replication. A load factor at this level of 85% rather than 15% can therefore be expected. Similarly, high level processes can avoid overwriting existing copies of files at medium levels. Thus one might end up with the amount m of replication increasing with the security level; determining optimal values will probably require an implementation.

Of course the lowest level password would be divulged at once by a user under compulsion; but in the absence of compulsion, the present construction can ensure that the lowest level in the system is not only secure but highly efficient. Although writing a file would be slightly slower than in a standard non-secure operating system, reading might well be quicker. In a typical application where the bulk of the data on a system is stored at the lowest level and only a small amount at higher levels, the performance penalty (in terms of both time and space) could be very low.

Thirdly, even when the disk is full (in the sense that a block has been corrupted, or that the capacity of any error correction code used has been exceeded), most of the user file content will still be recoverable; in particular the most recently written files are very likely to be intact. So the availability of files degrades gracefully, unlike in the previous construction.

Many further optimisations are possible. Even without a linear access hierarchy, we might still benefit from having a high performance lowest level; and we might define directories at higher levels using orthogonal families of codes rather than purely pseudorandom replication in order to increase efficiency. No doubt implementation will uncover many more optimisations: we leave that for future papers.

4 Relevance to Classical Steganography

The classical way of hiding information in a deniable way would be to use a steganographic program to embed the information in large files such as audio or video [3]. This has a number of problems. Firstly, one can only hide so much information, or its presence becomes noticeable; and if the opponent is allowed to subject these objects to small distortions of his choice, then the usable bandwidth can be very low indeed [8]. Secondly, our aim here has been to design a practical system, and that means one that can be implemented so that normal Unix or Windows applications might run on top of it. Command-line file recovery using a steganographic tool is inconvenient in such an environment; transparent operation is greatly preferable. Thirdly, given the assumption that the opponent knows the system, one might as well dispense with the cover objects and write the secret information to disk directly.

However, there may well be an interesting interaction between our second construction and the world of copyright marking. It is often a requirement that many copyright marks be superimposed; for example, the International Federation of the Phonographic Industry stipulated this in a recent call for proposals for audio marking schemes [9]. There are marking schemes which are linear (in the sense of adding signals) and can accommodate this easily. However, this linearity brings problems with it [7] and many researchers are working on nonlinear schemes. The problem now is that multiple marks can interfere with each other; and we do not want a copyright pirate to be able to remove the rightful owner's mark by adding several marks of his own.

The insight behind our construction suggests that in such environments an efficient strategy may be to add a number of independent marks to each work (or to each segment of a work that has to be separately marked), with each mark replicated about six times and forward error correction used between them. (Of course, each mark may be separately encrypted in order to prevent correlation attacks.) In this way, we expect a good trade-off between on the one hand having the bandwidth to add a number of independent marks, and on the other hand forcing a pirate who wishes to obliterate the marks (but who does not know the key) to add so many marks at random places that the resulting distortion will degrade or destroy the resale value of the object. Such robust marking is the subject of separate work.

5 Conclusions

We have presented a new protection mechanism, the steganographic file system. It is designed to give users a high degree of protection against coercion, in that they can plausibly deny the existence of whole directories of files on their hard disk, even against an opponent with complete access to the system and the resources to reverse engineer every chip and understand every byte of software. We presented two possible mechanisms, one inspired by the one-time pad and providing security exponential in its system parameters, and another based on the computational security of block ciphers. These mechanisms tackle an important, practical problem that has hitherto been ignored in the security literature.

Acknowledgement: We are grateful to Charalampos Manifavas for writing the simulation that determined the available load factors for our second construction.

References

1. "Measuring the Diversity of Random Number Generators", R Anderson, R Gibbens, C Jagger, F Kelly, M Roe, *preprint*, 1992
2. "Stretching the Limits of Steganography", RJ Anderson, in [3] pp 39–48
3. *'Information Hiding'*, May 30 – June 1 1996; proceedings published by Springer as Lecture Notes in Computer Science vol 1174
4. "Tamper Resistance — a Cautionary Note", RJ Anderson, MG Kuhn, in *Proceedings of the Second Usenix Workshop on Electronic Commerce* (Nov 96) pp 1–11
5. 'Plausible Deniability', DR Beaver, *Pragocrypt 96* pp 272–288
6. "Plug and Play Encryption", DR Beaver, in *Advances in Cryptology — Crypto 97*, Springer LNCS v 1294 pp 75–89
7. "Can invisible watermark resolve rightful ownerships?", S Craver, N Memon, BL Yeo, MM Yeung, Fifth Conference on Storage and Retrieval for Image and Video Database, 13–14 February 1997, San Jose, CA; *SPIE* vol 3022 pp 310–321
8. "Attacks on Copyright Marking Systems", FAP Petitcolas, RJ Anderson, MG Kuhn, *in these proceedings*; this paper is also available online at `http://www.cl.cam.ac.uk/~fapp2/papers/ih98-attacks/`
9. *'Request for Proposals — Embedded Signalling Systems'*, June 97, International Federation of the Phonographic Industry, 54 Regent Street, London W1R 5PJ

10. "File Organisation: Implementation of a Method Guaranteeing Retrieval in One Access", PÅ Larson, A Kajla, in *Communications of the ACM* v 27 no 7 (July 1984) pp 670–677
11. *'Cryptography and Evidence'*, M Roe, Cambridge University (PhD Thesis, 1997)
12. "Developer tortured by raiders with crowbars", M Weaver, *Daily Telegraph*, 31 October 97