Chameleon — A New Kind of Stream Cipher

Ross Anderson and Charalampos Manifavas

Cambridge University Computer Laboratory
Pembroke Street, Cambridge CB2 3QG, England
(rja14,cm213)@cl.cam.ac.uk

Abstract. Stream cipher systems are used to protect intellectual property in pay-TV and a number of other applications. In some of these, it would be convenient if a single ciphertext could be broadcast, and subscribers given slightly different deciphering keys that had the effect of producing slightly different plaintexts. In this way, a subscriber who illegally resold material licensed to him could be traced. Previously, such tracing could be done using a one-time pad, or with complicated key management schemes. In this paper we show how to endow any stream cipher with this potentially useful property. We also present a simple traitor tracing scheme based on random coding with which it can be used.

1 Introduction

The electronic distribution of intellectual property such as computer programs, clip art, databases, videos and music, often involves encryption followed by broadcast, with decryption keys being supplied out of band to subscribers who have paid for a particular object.

Computer programs and clip art are commonly distributed on CDs that contain extensive libraries, each item being typically encrypted using a different key. Customers purchase items by calling a service bureau and quoting a credit card number; a key is then read out to them over the phone. A number of firms sell encrypted databases: one is a compendium of building projects in certain counties of California, which is sold to building materials salesmen. Videos are broadcast encrypted on a number of satellite channels, and the decryption keys are sold to subscribers on smartcards.

A common problem with such systems is that some subscribers re-sell the information they have licensed. This is against the terms of their licence, and if they are detected they may be sued. Technical measures may also be used, such as failing to renew their encryption keys. However, given that the available technical measures are imperfect, with pay-TV pirates forging each successive generation of subscriber smartcard [5], and given that strong protection mechanisms are often in conflict with exportability and functionality, there is a shift towards combining technical protection with legal sanctions.

In any case, the important question is how cheaters can be detected.

One common approach is to customise the software as it is installed. Common techniques include inserting the licensee's name, giving a banner at the top of the screen stating something like 'This copy no. 123456 licensed to Bloggs the Butcher'. Another is to monitor the PC environment to detect re-installation, and a third is to have a timelock enforcing re-registration. However, all such mechanisms depend on 'security through obscurity' and can be broken by technically sophisticated pirates tampering with the software.

A second approach is to mark the information before it is encrypted. For example, a database supplier may mark each copy database in a unique way. Such 'fingerprints' have been in use for generations, having been used to mark mathematical tables and other early instances of intellectual property. (For a survey of fingerprinting, see [12].)

If manufacturing a unique database for each customer is too expensive, as it might be if the database is shipped initially on a CD-ROM, the supplier can use other techniques. For example, if he sends out a weekly update to subscribers, he can produce two different versions that differ slightly. By sending these two different versions to different partitions of his N subscribers in successive weeks, he can track down the cheater in $\log N$ weeks.

Whatever strategy is used to mark individual copies of the information, an attacker can always purchase a number of copies and compare them. Nonetheless, not all attackers are well organised, and it is often thought worthwhile to have mechanisms that ensure a certain minimum number of copies will have to be purchased. Matters can be arranged so that any captured pirate copy will correctly identify the subscriber who deciphered it, or — if up to a certain number of subscribers collude — it will correctly identify at least one of them, and will not mistakenly identify any innocent subscribers. This is known as 'Traitor Tracing' [6] and we will return to it below.

Several problems remain to be solved. Firstly, broadcasting more than one ciphertext is expensive and in many applications (such as satellite TV) it is impractical. So we may want there to be only one version of the ciphertext. Secondly, if we rely on software to insert the user's identity on decryption, then it is likely to be disassembled and interfered with by pirates. Even if we use 'trusted' hardware, this will be expensive and may be ultimately vulnerable to attack [5].

So we want a scheme that will enable us to give different keys to different subscribers, in such a way that they decrypt a single broadcast ciphertext in different ways.

The approach taken by [6] and a number of subsequent workers is to mark not the plaintext but a 'virtual key'. This decryption key is computed from a number of user keys; each user gets a sufficient but unique set of these keys, and matters are arranged so that a certain minimum number of users need to collude to construct a key that works but identifies none of them. One problem with this approach is that the bandwidth required for the control messages may not always be available.

If we could use a one-time pad, then we could just well each user a slightly different deciphering key, and they would end up with slightly different plaintexts. However, in applications such as the distribution of videos and music — where such a scheme would be most valuable — the amount of key material required would be prohibitive.

So it would be useful to have an encryption algorithm with the property that a slight change in the key will result in a slight change to the plaintext that is deciphered from a given ciphertext.

One might think that this would expose the cipher to divide-and-conquer attacks, as an attacker would be able to tell when a guess of the key was 'almost right'. But we show that this is not necessarily so. Any stream cipher can be modified simply so that a slight change in the key will cause a slight change to the output keystream. Yet, in practical cases of interest, the construction appears to strengthen rather than weaken the cipher.

2 The Construction

Our construction can be concisely described by a concrete example. We take a conventional pseudorandom generator (which in our prototype is the block cipher that forms the core of the 'Tiger' hash function, run in output feedback mode, rather than in feedforward mode as in the hash function) [4]. The particular choice is unimportant for our construction — we could as easily use any block cipher in output feedback mode, or a dedicated stream cipher such as PIKE [3]. The key for this stream cipher we will call key 'A'.

Next, we take a table of 2^{16} 64-bit words — 512 KB of random data — which we call key 'B'.

In order to encipher a 64-bit word of plaintext we take a 64 bit word from the keystream generator and use it to select four words from key 'B', which we exclusive or together. The result is the keystream; it is exclusive or'ed with the plaintext to get the ciphertext (and, when deciphering, with the ciphertext to get the plaintext).

The effect of a one-bit change in key 'B' is to change about 4 bits per 512KB of keystream generated. These changes are at the same locations in the word as those in the key; thus, when enciphering audio signals that have been digitised into 16-bit words, we can arrange that the copyright marks appear in the least significant bits.

3 Tracing Traitors

A common concern with systems that give intellectual property a unique mark for each subscriber is that a pirate may purchase, say, three copies of a work in different false names and then obtain an unmarked copy by using bitwise majority voting.

There are a number of strategies available to make such attacks more difficult. The basic idea is that for some small integer k, a pirate plaintext (or decoding device) should disclose the identity of at least one of up to k copyright violators who pooled their plaintexts (or secret keys), and that it should not be possible for an innocent subscriber to be framed [6].

These techniques give only lightweight protection in that they are effective only for small values of k. Indeed, Shamir has pointed out that these 'traitor tracing' schemes suffer from the problem that as k increases, the defender does exponentially more work in order to cost the attacker linearly more effort [11]. However there is usually little point in trying to guard against a large conspiracy, as an attacker who could organise it could also manage to subscribe in a false name.

So the realistic goal of traitor tracing is to provide a pragmatic defence against unsophisticated attackers, and in this spirit we offer a simpler way of implementing it than [6]. Our technique was inspired by [7].

In the concrete system given in the above section, with four lookups into a table of 4 megabits, assume that there are 4000 marked bits. Thus, as somewhat over the square root of the total number of bits are marked, we expect that any two users will have a marked bit in common, and that these common bits will be unique to each pair of users. Thus if any two subscribers collude, they will succeed in eliminating all but one of the marks from their 'B' keys, but the remaining mark (or its effects on the plaintext) will identify them.

So if three users collude and attempt to produce a clean copy by bitwise majority voting, the resulting text (or B key) will still incriminate each of them, two at a time, with high probability. Even if four users collude, they can identify the incriminating marks, but not figure out how to remove them. Thus our random coding approach gives us a simple traitor tracing scheme with $k = 4$.

How practical is this? Take for example an audio marking scheme. With 16 bit encoded uncompressed audio, we might want to limit the marks to the least significant bit of each 16-bit word. Thus the number of effective bits in the 'B' key is only 256K, so we need mark at most 1000 of them. This leads to the marking of 1.6% of the least significant bits, which is unnoticeable for most modern music. We will discuss an approach for video signals below.

More complicated marking schemes can be devised (e.g. [9, 10]) and used with our scheme. Our construction is independent of whether the marks on the 'B' key are randomly or systematically generated; the changes they induce in the keystream not only preserve the bit position in the word, but also incidence structures, which is what we generally need for traitor tracing schemes to work.

As with the somewhat different construction of [6], there is no need to penetrate the tamper resistance of a captured pirate decoder. Its behaviour is quite sufficient to identify the subscribers whose keys were used to construct it, assuming that this can be done at all.

4 Key Management

The 'A' keys are quite conventional and can be managed using the conventional machinery of crypto protocols. For example, the current mechanism in several pay-TV systems is to compute a working key as a MAC of all the control packets that have been transmitted in the previous time period. This is so that once a traitor (such as a cloned subscriber card) has been identified, a packet can be sent in each time period instructing that card to commit suicide. If a user blocks this instruction to prevent it reaching his smartcard, then this card cannot calculate the current key and the cloned card is thereby rendered useless. Such key management techniques can be adopted unchanged in the system proposed here.

Managing the 'B' key is more difficult. One might simply treat it as a long term key installed by out-of-band means; if it is used, together with a suitable 'A' key, to generate a lower level 'B' key, then this will have about four times as many marks in it as the long term key did. The possible advantage of having master and session 'B' keys is that re-keying might help discriminate between candidate conspiracies with a higher probability than otherwise. The exact probabilities, and thus the advantage if any, would depend on the parameters of a given application.

5 Performance

The performance degradation is not large, so long as the 'B' key remains in memory. This is the most critical parameter and it can be tuned to the equipment in use.

If the underlying pseudorandom generator is triple DES, then it is unlikely that our construction will add a significant penalty. Even if the generator is a high speed software algorithm, the penalty is not enormous. For example, when we use Tiger, running in output feedback mode on a 275MHz Alpha workstation, we can generate raw pseudorandom bits at 67 Mbps; and when using four lookups to a 512KB table, we still get 42 Mbps. We expect that this can be improved by careful optimisation.

In audio applications, performance is unlikely to be a problem; we can decrypt a minute's worth of music in about a second. Performance is only likely to be an issue in applications such as video, and especially where MPEG decoding places a high load on the processor. In such applications, a bit error rate lower than 0.1% may also be required; so the pragmatic approach is to mark only a subset of the content. One might for example process only one block in a hundred using our construction, and select this block using the native mode stream cipher (which would also be used to encipher the rest of the content).

By using higher density marks, one can construct schemes that are 6-resilient, 8-resilient and so on. The higher density of marking can be offset by marking a smaller subset of the content; however, the comments of section 3 still apply, and there is the further problem that if the marks are made too dense in any subset of the content whose selection is independent of the 'B' key, then an attacker might replace this subset with completely random noise.

6 Security

Despite the poor diffusion of the extra key material, our construction appears to make it more difficult to attack the underlying generator. We can distinguish two cases: the outside attacker (who does not know the value of the 'B' key at all), and the recently revoked insider (who knows all or most of it).

Where the 'B' key is unknown, then it seems that even a very weak generator may resist attack. For example, if we use the multiplexer generator or the nonlinear filter generator, the known attacks [1, 2] do not work.

The more realistic attack scenario is that the attacker knows the 'B' key — or most of it. In this case, attacks are still harder, as there is often equivocation about the pseudorandom input to the tables that generate a given keystream output (and the mapping between the pseudorandom generator output and the keystream is a bit too large to store in any case). The details will be a function of the table size and the number of values that are taken from it; but in general, the effect of the table lookup is similar to that of applying a known pseudorandom function to the generator's output.

Our construction may induce some degenerate behaviour that did not exist before. For example, when we use four lookups to a 512K table, we will get a zero keystream whenever the input pseudorandom value is of the form *abab*, *aabb* or *baab*. This weakness does not arise when using three lookups into an 8 MB table, but in that case we are using only 60 bits of pseudorandomness to generate 64 bits of keystream. But, as far as we can see, these weaknesses are of no practical use to an attacker in the kind of applications in which we envisage our construction being used.

A further security advantage of our construction is that the keys are very much larger than in conventional cryptosystems — hundreds of kilobytes, or megabytes, rather than tens of bytes. In fact, the work that led to this construction was inspired by a realisation that in the modern world, many of the most potent threats to cryptographic security involve either malicious code or attacks over networks; in this environment, big keys are good because they are harder for a virus or network intruder to steal without being detected. Another inspiration for this work was [8], which also uses table lookup and xor to construct a stream cipher, but for a completely different purpose.

Finally, the following attack was suggested from the floor at the workshop: colluders, having removed almost all of the marks, could then insert a number of random marks to provide camouflage and hopefully frame other users. Thus, with the concrete example given above, the three attackers might add another 4000 bits to the three remaining genuine marks. However, the three genuine marks can be detected as they form a 'triangle' joining the three conspirators together.

This attack does force the publisher to examine 4,000 marks rather than three; but, as we have emphasised, a serious attacker would simply organise a sufficient coalition — or subscribe in a false name.

7 Conclusion

We have shown how to do two seemingly contradictory things to an arbitrary stream cipher — to strengthen it, and to endow it with the property that small changes in the key cause only small changes in the keystream. We name this construction 'Chameleon'.

Like other traitor tracing schemes, it has only limited collusion resistance. However, we believe that there are applications in which it could be useful. In addition, as it embodies a new kind of cryptographic mechanism, we hope that it will inspire other new work — whether better fingerprinting schemes, or new applications entirely.

Acknowledgement: This work was carried out as part of the joint EPSRC/DTI funded project 'NetCard', which supported the second author. We also acknowledge the referees whose comments enabled us to improve the presentation.

References

1. "Fast Attack on Certain Stream Ciphers", *Electronics Letters* vol 29 (22 July 93) pp 1322–1323
2. "Searching for the Optimum Correlation Attack", in *'Fast Software Encryption'* (1994), Springer LNCS vol 1008 pp 137–143
3. "On Fibonacci Keystream Generators", RJ Anderson, in *Fast Software Encryption* (1994) Springer LNCS vol 1008 pp 346–352
4. "Tiger: A Fast New Hash Function", RJ Anderson, E Biham, in *Fast Software Encryption* (1996), Springer LNCS vol 1039 pp 89–97
5. RJ Anderson, MG Kuhn, "Tamper Resistance — A Cautionary Note", in *Proceedings of the Second Usenix Electronic Commerce Workshop* (Nov 1996) pp 1–21
6. "Tracing Traitors", B Chor, A Fiat, M Naor, in *Advances in Cryptology — Crypto 94*, Springer LNCS vol 839 pp 257–270
7. "On Key Storage in Secure Networks", M Dyer, T Fenner, A Frieze, A Thomason, in *Journal of Cryptology* v 8 no 4 (Autumn 95) pp 189–200
8. "Conditionally-perfect secrecy and a provably-secure randomized cipher", U Maurer, in *Journal of Cryptology* v 5. no 1 pp 53–66
9. "Asymmetric Fingerprinting", B Pfitzmann, M Schunter, in *Advances in Cryptology — Eurocrypt 96*, Springer LNCS vol 1070 pp 84–95
10. *'Anonymous Fingerprinting'*, B Pfitzmann, M Waidner, IBM Research Report RZ 2881 (#90829) 11/18/96, IBM Research Division, Zurich
11. A Shamir, *comment made from the floor of the conference*
12. "Fingerprinting", in *Proceedings of the 1983 IEEE Symposium on Security and Privacy* pp 18–22