Cryptographic attacks

By Prof. John Larmouth

1 Introduction

This note is purely concerned with attacks against **conventional** (symmetric) encryption, designed to support the non-disclosure function.

It is largely a tutorial, and should not be read by anyone seriously into cryptography!

It makes the point that it is possible to prescribe a cryptographic function for non-disclosure (that can be applied to a CXER encoding) that is more secure than functions that can be applied to a general XML encoding (specified by XSD or by a DTD).

It also makes the point that using ZIP to "randomise" an XML document (and to remove explicitly visible patterns from it), provides only a very minimal enhancement of security, compared with the application of the function described above to CXER.

2 The model for attacks on non-disclosure algorithms

The conventional model is of an encryption device that takes two inputs - a secret key and a clear message. It outputs an encrypted message.

At the receiving end, a similar device takes the encrypted message as input together with the same secret key and reproduces the clear message.

Attacks are aimed at discovering (using, typically, a number of transmitted encrypted messages) the secret key that is used.

In all cases (for theoretical analysis) it is assumed that the algorithms employed by, and the operation of, the encryption devices are known. Only the secret key is assumed to be secret.

The underlying assumption here is of a super-computer randomly (by sequential trial and error) trying to crack the code, that is, to find the secret key being used, knowing fully the algorithms employed in the encryption process.

The use of (human) agents to steal codebooks or to obtain keys by bribery or by physical intrusion on a smart card, etc. is not considered in the model.

Attacks (to find the key, and hence to decode other messages) are categorised as follows:

- Chosen clear message attacks: In this case, the attacker has obtained an encode-only code machine or smart card with a code already in it. Clear messages can be fed in, and the output examined. The aim is to determine the secret key by comparing the clear input messages with the encrypted output messages.
- Known clear message attacks: In this case the encrypted message is available for analysis, and all of (or more commonly some part of) the clear message is known to contain specific text (bit strings). In the simplest case, the entire clear message is known.
- Known clear message pattern attacks: In this case, it is known that all clear messages contain some pattern. This may be English language words, or encodings of some non-compact alphabet such as ASCII letters, or ASN.1 BER tags and lengths, or XML start and end tags, or some combination of all of these.
- There is no known pattern of bits in the clear text messages. All possible patterns of bits can represent possible messages. In this case the code cannot be cracked by brute force methods.

This model and classification is fundamental to any discussion of vulnerability of non-disclosure cryptographic algorithms.

The focus in this document is on "known clear message pattern" attacks. This is what good communications protocols should guard against. Other forms of attack are not relevant to the discussion of communication's protocols, but relate to physical security issues or to cryptographic algorithm issues.

A simplistic (but adequate for many purposes) way of looking at these attacks is to consider a super-computer programmed to decrypt a message trying all possible keys one after the other, and flagging for further analysis keys where the clear message produced looks suspicious - contains the known clear text, or the known patterns.

An encoding of information and a non-disclosure encryption function is good if the super-computer will produce a vast number of potential keys - keys that generate the known patterns in the decrypted messages.

3 The problem with different encoding rules

In general, if any possible bit-pattern in a message has no redundancy, and corresponds to some possible real message, then we have a very secure encoding: There are no patterns, and no redundancy.

If we ignore possible redundancy due to poor application-level design (use of character strings at the application level), PER encodings are very good.

Almost any possible bit-string, fed to a PER decoder, will produce an abstract value for the application. This makes PER a very good starting point for a non-disclosure encryption transformation.

A BER encoding has much more redundancy, and hence has more vulnerability to known pattern attack.

An XML encoding has very great vulnerability, due to the presence in the clear text of:

a) The encoding of opening and closing angle brackets;

b) The use of ASCII text - usually English language - encodings, which use only a small number of the possible bit patterns of an octet;

c) The occurrence of start-tags and end-tags that are identical encodings over several tens of bits.

4 The uselessness of ZIPping

It is unfortunately a common view that if the bit-pattern of a message with a known pattern is randomised, for example by applying the .ZIP function, before encryption, then the known pattern attack is eliminated.

This is not true, as a little analysis shows. The conversion from a ZIPped file to the original file is totally algorithmic, known, and deterministic (one-to-one).

The super-computer doing the code breaking simply unzips before looking for the known patterns. (Redundancies in the ZIP format can short-circuit this check.) All that is gained is a small increase in the time for each trial (an unzip process has to be added at each attempt).

There is no significant gain in security if the original material has known patterns.

This point needs to be understand, and is often **not** understood by laymen discussing security issues, particularly those concerned with security of XML transfers.

5 Why can a cryptographic transform of CXER work?

CXER is XML, and has known patterns. Indeed, it has known clear text (the tags), assuming the schema is not secret. So how can we produce a robust non-disclosure cryptographic transform for it?

NOTE – As stated earlier, patterns (e.g. clear text) at the abstract - application - level are not of concern here - the application designer guards against these.

The approach is well known: we need to eliminate redundancy, and patterns, as part of the cryptographic algorithm. This requires that the cryptographic algorithm has knowledge of the patterns to be eliminated.

This is not just a case of saying "remove the XML tags" - some tags, such as those that determine presence of optional elements or choices, carry application semantics. These tags can be reduced to one or two bits, but cannot be eliminated. Doing such removal on an ad hoc basis is almost impossible in the general case.

However, taking a CXER encoding (XML), and converting it to the corresponding abstract value, then encoding with CPER, we get a bit pattern that can then be encrypted, and which contains very few known patterns. (Almost all PER encodings correspond to one of the messages the application is able to send.)

How is this different from using ZIP? This is a key point.

Consider again our super-computer. It tries a key. It decodes with PER (against a known schema). That gets an abstract value (almost certainly, in all cases - but, of course, not the correct one). It encodes that with CXER. OK. The pattern is there - for every key it tries!

This non-disclosure encryption transformation is **not** vulnerable to known pattern attack, despite the fact that the original CXER had known patterns in it.

6 How is this magic being worked?

The point here is that what we are really encrypting is not the XML that has the patterns and redundancy in it, but the underlying abstract values.

These may have patterns and redundancy if the application has been badly designed, but that is an application matter, and not a fault of the encoding that is being used.

7 Summary

It is well known that the sequence: XML, ASN.1 BER, ASN.1 PER produces immense gains in the bandwidth needed for the communication. That is not relevant in this document.

ZIPping generally improves all three encodings for bandwidth, but typically in the same ratio, because of Shannon's Information Theory.

It is also the case that encryption of the same sequence of encodings produces increasing gains in vulnerability against non-disclosure as you proceed through the sequence. This was the main issue described in this paper. Again, ZIPping produces little or no improvement in security vulnerability.

Finally, it is important to note that we **can** encrypt XML material (human readable, well-formed XML) for non-disclosure **without** security vulnerability, provided ...

... the schema is known, and is known to be an ASN.1 definition.

The conversion to an abstract value and then to a PER encoding is part of the cryptographic non-disclosure function. **The clear message, sent from the originator and delivered to the recipient remains XML.**

JL

9 April 2002