MEO - Multiple Eyepairs Only*

Peter Stamfest

November 11, 2012

Abstract

The use of public key cryptography to encrypt information intended to be only accessible by a group of people is explored. A practical implementation of the presented system is introduced which can use RSA keys and keypairs in the form of X509 certificates and PKCS12 keystores or hardware crypto tokens accessible using a PKCS11 provider. The corresponding file format is designed to reveal as little information about the participants as possible even to other participants.

1 Introduction

There are often situations where one wants to encrypt information to be accessed by other persons. In practice this is often done by encrypting information for a single recipient using public key cryptography (eg. by using the RSA algorithm). However, some information is more sensitive, in that it would be convenient to not allow a single person to access the information but rather a group of people, maybe employing a m out of n approach (that is any number of $m < n$ people should together be able to access the information, but any number p of people smaller than m should not be able to.

This article defines an algorithm to achieve this goal. It also defines a file format to store the required information and points to an open source implementation of the algorithm in the form of a commandline tool to create such files and a graphical user interface to decrypt them.

The author is fully aware that this is most likely not original research, but so far he has never come across a good explanation or an easily available implementation of the algorithm. He also strongly believes that there are applications for such a system in various areas, however this paper will not go into details about this. He also believes that a public implementation could be put to good use in various circumstances.

1.1 A word of caution

The presented algorithm and its implementation may be seen by some as being "dangerous" by providing means of communication for "bad guys" previously unheard of. The author is fully aware of possible negative uses of such a technology. However, there is nothing really new here: "Bad guys" already have the possibility to achieve similar goals by simply encrypting information multiple times. It is the "Good guys" that miss simple tools to use such a technology for "Good reasons".

*Please note the pun on eyepairs vs. keypairs

2 The algorithm

2.1 Preconditions, basic desired properties

It is desired to encrypt a potentially large amount of data. This is done by using a fast symmetric cipher for the bulk data. The used symmetric key (and potentially an initialization vector, IV) are to be stored with the data in a secure way.

A group of public keys (of size n) is available (one per key-holder) to be used to encrypt the symmetric key in a way that allows a number $m < n$ to access it in order to decrypt the information again.

It is also desired that a group of key-holders of size $k < m$ should not have an advantage with respect to brute-force breaking of the used symmetric cipher in comparison with the direct breaking of cipher without any prior knowledge of the used key. This puts constrainst the algorithm.

From now on the holder of keypairs are called "attendees" for the scope of this article.

2.2 Basic operation

We define a number of different, purely random sequences of bytes (called "parts", denoted as P_i for the i-th part.), each of which is the same size than the required session key. The session key S becomes the XOR of all those byte sequences:

$$S = P_1 \otimes P_2 \cdots \otimes P_p \tag{1}$$

This has the property that every single part is required in order to obtain the session key, because every part can change every bit of the session key. It also means that every part is of equal importance and that trying to brute-force one part is as hard as brute-forcing the session key itself. Thus all properties of the used session cipher remain intact.

Effectively, every part forms a one-time-pad to encrypt the session key. All other parts XORed together are the cipher text of the plain text version of the session key with respect to that part.

Parts get distributed to the attendees in a way that each attendee receives only a fraction of all parts. An attendee may receive multiple parts. Every part gets encrypted using the public key of the attendee, so that only the attendee is able to decrypt it back into the original part.

2.3 Alternative approach

An alternative would be to apply p different encryption stages with each stage using one of the parts as its key. Every stage might even use a different symmetric encrytion algorithm. This might improve overall security[1], but also increases processing time considerably. In this case the symbol $\otimes$ would indicate a sequence of encryptions and most of the discussion of the algorithm still holds.

Decryption would have to reverse the sequence of ciphers used in the stages, of course.

2.4 Number and distribution of parts

The number of parts to generate and how to distribute parts to attendees depends on n and m only and is a question of combinatorics only:

In order to require m out of n people we observe that $m - 1$ people should not have access to all parts. If we decide that missing one part is enough, then we can

[1] Cryptographer, anywhere?

Attendee						
1	2	3	4	5	Part number	thus known by attendees
x	x	-	-	-	1	3,4,5
x	-	x	-	-	2	2,4,5
x	-	-	x	-	3	2,3,5
x	-	-	-	x	4	2,3,4
-	x	x	-	-	5	1,4,5
-	x	-	x	-	6	1,3,5
-	x	-	-	x	7	1,3,4
-	-	x	x	-	8	1,2,5
-	-	x	-	x	9	1,2,4
-	-	-	x	x	10	1,2,3

Table 1: Example 1 - Part distribution if 3 out of 5 attendees should be required.

assign one part for every such combination *that only all the others have access to*. This means that for every combination of $m-1$ attendees at least one part will be missing and this part is known by any one of the others.

The total number of required parts p is thus given by the binomial coefficient:

$$p = \binom{n}{m-1} = \frac{n!}{(m-1)!(n-m+1)!} \tag{2}$$

This scheme guarantees that m attendees will always have access to all the parts and they can thus reconstruct the session key.

It works for all combinations $1 \leq m \leq n$. For the case $n = m$ it degenerates into one distinct part per attendee. In the case $m = 1$ for any n the part is the session key and every attendee has direct access to it.

2.4.1 Example 1

3 out of 5 should be able to access information.

This means $\binom{5}{2} = \frac{5!}{2!3!} = 10$ combinations of people missing one part are possible. These are shown in table 1: ("x" means attendee takes part, "-" means the attendee is not taking part)

3 Defining a file-format

This section defines the file format for "MEO" files. A proposed file extension for such files is ".meo", although this might be seen as already revealing too much information.

The following design principles were employed when defining the file format:

- To create a MEO file it must be sufficient to have access to the public key of all involved attendees only.
- To allow for a maximum of anonymity for attendees it is desired to make it as hard as possible for an attacker to even find out which public keys were used to create the file.
- The file format should be extensible.
- As few meta data as possible should be revealed by a MEO file. One meta data item that was assumed to be of only minor importance is the length of

Byte offset	Length	Content
0	2	Length l of data-part (payload) of the record in network byte order. This essentially is the block-size of the RSA key used to encrypt it. Due to the used padding the decrypted payload data might be shorter (denoted as l')
2	2	a random clear-text "index" of the record. The index is used within the payload to refer to the next record for the attendee.
4	l	An RSA encrypted payload. It uses PKCS#1 padding.

Table 2: Record structure of a MEO header record

the data to be encrypted. If this is of concern, actions can be taken on the unencrypted data prior to creation of a MEO file.

- RSA-key operations should be possible on common crypto hardware / tokens.
- The number of required RSA operations should be as small as possible, because crypto hardware is often rather slow in practice.

Based on these principles a vast range of different file formats would be possible. The file format described in the next section is thus not the only one possible, it is also not claimed to be the best one.

3.1 The file format explained

The file format consists of a header and a data section. The data section immediately follows the header section and is just the cipher text of the chosen symmetric cipher algorithm using the session key and session initialization vector applied to the input data.

The header section consists of a sequence of potentially differently sized records. Each record consists of the information layed out in table 3.1. Every record holds a "payload".

Numeric multi-byte information is stored in network-byte-order both in- and outside of the payload data.

The end of the header is specified by a record having record length 65535 (that is 0xffff in hex) and a meaningless random index field. The byte after that final index is the first byte of the data part.

Currently only two payload record types are defined. The payload record type can only be found-out after the successful RSA decryption of the part. Because the key used to encrypt the payload record is not indicated outside of it, an attendee must (in principle) try to decrypt all records and check if it contains information for him.

An attendee can decide if any given record is for him/her based on two different kinds of information obtained through or after the decryption attempt:

1. The used PKCS#1 padding: If there is no padding error, it is extremely likely that the record is for the attendee attempting the decryption.
2. Every payload record contains a SHA-1 MAC (message authentication code) of the payload information at its end.

The use of the clear-text index shown in the record format allows to reduce the number of required RSA operations to check/decrypt records, because every

Byte offset	Length	Content
0	1	The ASCII character 'M'
1	2	The random clear-text "index" of the next record intended for the attendee.
3	j	The name of the symmetric cipher used for the data part. This field is NUL terminated. The cipher naming follows the scheme used by the openssl crypto library. The standard cipher used is "aes-256-cbc". This information defines the length of the symmetric key S_{len} and the corresponding IV IV_{len}. The sum of these lengths defines the length of the secret parts.
$3+j$	2	part-index i (zero-based)
$5+j$	2	Total number of attendees used to encrypt the data part of this MEO file
$7+j$	2	Number of attendees required to decrypt the data part
$9+j$	2	You are attendee number x
$11+j$	$l'-11-20$	random padding
$l'-20$	20	SHA-1 MAC of the first $l'-20$ bytes of this record.

Table 3: "M"-payload record structure for a l' bytes long payload

payload record contains the index of the next record intended for the attendee once the first record for the attendee has been found.

The "M" record type defines metadata intended for the attendee. The record type is defined as shown in table 3.1.

The "P" record type defines the part data for a part known to the attendee. The record type is defined as shown in table 3.1.

Currently, all M-records come before all P-records. This is in order to reduce the number of RSA operations needed to find the proper "index-chain" for a given attendee.

For increased anonymity there is, on purpose, no official "magic number" identifying the file format, although MEO files often start with one of the byte sequences "0x00 0x80", "0x01 0x00" or "0x02 0x00" due to commonly used RSA keypair sizes.

Byte offset	Length	Content
0	1	The ASCII character 'P'
1	2	The random clear-text "index" of the next record intended for the attendee.
3	2	The 0-based part number contained in this payload record
5	1	$S_{len}+IV_{len}$ - for redundancy
6	$S_{len}+IV_{len}$	the random part used to construct the symmetric key and the IV for the session cipher
$6+S_{len}+IV_{len}$	$l'-6-S_{len}-IV_{len}$	random padding
$l'-20$	20	SHA-1 MAC of the first $l'-20$ bytes of this record.

Table 4: "P"-payload record structure for a l' bytes long payload

Also, there is a characteristic pattern of records sizes throughout the header part of MEO files due to the nature of the header as an effective "linked list" of records.

3.2 Planned extensions

The file format is extensible by design (by defining new payload record types). It is planned to optionally include some more meta data about either the attendees or the data. This could include:

- The list of other attendees required to decrypt the data part (accessible by attendees only)
- The filename / timestamp of the data part (accessible only under the same circumstances as the data itself)

Every such extension would have to be evaluated regarding its impact on security.

4 Security properties / questions

As the author is not a cryptographer himself all statements in this section should be viewed as being preliminary. Review by cryptographers would be welcomed.

These are the properties and open questions for both the algorithm and the file format:

- The protection of the secret information is as good as the used symmetric cipher. This is because all the parts are as long as the symmetric key and every part can change all bits of the key.
- The use of the presented "part-distribution" scheme also does not impact the security of the symmetric cipher.
- An attacker trying to crack an MEO file would most likely do so by attacking the data part directly, because there is only one encryption step involved. The alternative scheme mentioned shortly in section 2.3 would allow to mitigate this, especially if different symmetric cipher algorithms would be used in each step[2].
- Question: is the record and payload data format (including the used PKCS#1 padding) safe with respect to any known RSA weaknesses? There is some redundant and/or purely informational data there that is not strictly required and that might be used for known-plaintext attacks.

5 Open-Source implementation

There is an open source implementation of the algorithm available. A command-line tool can be used to create MEO files given a list of X.509 certificates and the input data. A GUI (graphical user interface) can be used to decrypt such files. The GUI allows to use PKCS#15 keystores or crypto-tokens using a PKCS#11 provider. The GUI handles all aspects of decryption, including attaching tokens in sequence.

This implementation currently does not implement the alternative approach of applying a number of different encryption stages, because one of its envisioned modes of operation is to encrypt even high volume data in real time.

[2]This stronly depends on any possible group and/or commutation properties the used ciphers might or might not have, but I am quite sure that for commonly used cipher algorithms there is no such structure among each other that would allow an attacker advantages, so the alternative scheme might actually improve security considerably.

6 About the author

Peter Stamfest graduated from the University of Technology in Vienna with a master-level degree in physics in 1998. He started to work as a self employed consultant in the areas of computer system design and administration, network and security design and administration in 1999. His interests include, among others, all forms of security-related questions, system monitoring and long-term operation.