File Encryption: Come e Perché

Una esigenza che alcuni hanno, e molti di più dovrebbero avere, è di proteggere i documenti presenti sul proprio elaboratore. Ad esempio molti portatili hanno dischi estraibili e se si viaggia molto vi è anche il rischio di perdere un disco, oltre a quello del furto dell'intero portatile. Se i dati sono importanti, una possibile soluzione per proteggersi anche in caso di furto o smarrimento dell'hardware è quella della cifratura dei documenti, in modo che colui che si trovi in possesso dell'hardware non sia in grado di comprenderne i contenuti senza conoscere la chiave crittografica. Cerchiamo allora un modo facile per farlo.

1. Crittare documenti con PGP

Uno dei modi più semplici è di usare PGP. Per rendere l'idea del processo usando del software disponibile a tutti, useremo qui, in ambiente Unix e dalla riga di comando,[1] GnuPG (GNU Privacy Guard) che è conforme allo standard OpenPGP descritto nel RFC2440. Dopo avere installato GnuPG, comunemente detto `gpg`, con le usuali precauzioni di controllare gli hash e le firme digitali, il comando `gpg --version` dà alcune informazioni sugli algoritmi di cifratura supportati (vedi ad esempio Tabella 1). Vogliamo ora cifrare il file `document.doc` con AES-256 e per far ciò diamo il comando

```
> gpg -c --cipher-algo RIJNDAEL256 -v -o document.enc document.doc
```

`gpg` ci chiede di indicare una passphrase che viene usata come chiave per AES-256. L'opzione `-c` indica che vogliamo usare solo un cipher simmetrico che è specificato dall'opzione seguente. Viceversa, il comando

```
> gpg --decrypt document.enc -o document_new.doc
```

chiede la passphrase e decifra il documento in `document_new.doc`.

Questo metodo non è però molto comodo poiché richiede di creare una passphrase per ogni documento, cosa che alla fine risulta poco pratica. Per evitare di dimenticarsi la passphrase, che ovviamente bisogna tenere a mente, si finisce ad usare la stessa passphrase per ogni documento, ma questo va evitato ad ogni costo. E' però possibile utilizzare uno schema più complesso che permette di usare una sola passphrase. Per motivi di sicurezza dovremmo utilizzare una Smart-Card, ma per semplicità illustrativa ci dotiamo invece di due buoni vecchi floppy, inseriamo il primo e montiamolo ad esempio sulla cartella `/mnt/floppy`. Creiamo una coppia di chiavi con un algoritmo asimmetrico (una chiave pubblica ed una privata) dando il comando[2]

1 L'utilizzo da interfaccia grafica è ancora più semplice, ma non ci è possibile riportare in questa sede le schermate.
2 In Tabella 2 è descritto il dialogo per la creazione delle chiavi.

```
> gpg --homedir /mnt/floppy --gen-key
```

(le opzioni `--no-options --lock-never` possono essere utili in questo e tutti i comandi seguenti). La chiave privata è protetta da una passphrase che ci viene richiesta. Il seguente comando ci mostra le chiavi che sono appena state create

```
> gpg --homedir /mnt/floppy --list-sigs
/mnt/floppy/pubring.gpg
------------------------
pub  1024D/172BC5C4 2003-01-28 Mister A (Dr.) <misterA@email.it>
sig        172BC5C4 2003-01-28 Mister A (Dr.) <misterA@email.it>
sub  2048g/F5D22B33 2003-01-28
sig        172BC5C4 2003-01-28 Mister A (Dr.) <misterA@email.it>
```

Copiamo ora il contenuto del floppy sul secondo floppy che metteremo in un posto molto sicuro come backup.

Usando il floppy così preparato possiamo cifrare il nostro documento con il comando

```
gpg --homedir /mnt/floppy -e --cipher-algo RIJNDAEL256 -v \
    -r 172BC5C4 -o document.enc document.doc
```

E decifrarlo con il comando

```
gpg --homedir /mnt/floppy --decrypt document.enc -o document_new.doc
```

il quale ci richiede la passphrase indicata al momento della creazione della chiave privata.

Una spiegazione è d'obbligo. Abbiamo preparato un floppy su cui ci sono una chiave pubblica ed una chiave privata, quest'ultima cifrata con la passphrase che abbiamo dato. Quando cifriamo un documento, `gpg` lo cifra con AES-256 con una chiave simmetrica arbitraria creata al momento, poi cifra questa chiave simmetrica con la chiave pubblica presente sul floppy e la appende al documento cifrato. Per decifrare il documento dobbiamo usare la passphrase per decifrare la chiave privata sul floppy, la chiave privata è poi usata da `gpg` per decifrare la chiave simmetrica appesa al documento, chiave che è usata a sua volta per decifrare il documento. In questo modo otteniamo che ogni documento è cifrato con una chiave diversa, ma noi abbiamo bisogno solo di una passphrase e del floppy per la cifratura/decifratura. A seconda degli algoritmi di cifratura utilizzati può essere praticamente impossibile per un malintenzionato, anche in possesso del floppy ma ignaro della passphrase, decifrare qualunque documento. L'uso del floppy è solo per aumentare la sicurezza rispetto a tenere le chiavi `gpg` sul disco rigido insieme ai documenti, ma è consigliato utilizzare una smart-card o simile device per maggiore sicurezza.

2. Ma quanto è sicuro?

Purtroppo questo approccio ha dei problemi. Ne consideriamo qui alcuni ignorando tutte le

problematiche di sicurezza relative al momento della creazione del documento, ad esempio se qualcuno sta registrando le nostre battute sulla tastiera e così via. Per accedere al documento dobbiamo sempre crearne una copia non cifrata, sia su disco o anche solo in memoria[3]. Dobbiamo perciò essere sicuri che non vi sia nessuno sulla macchina che possa catturare il documento in questo momento. Il problema è che una volta che il documento è stato in forma non cifrata nella macchina, non è così facile eliminarlo. Per eliminarlo dalla memoria RAM possiamo, se è il nostro portatile, spegnere, disconnettere la tensione, riconnettere e riaccendere la macchina. Dopodichè usiamo uno dei vari programmi disponibili che cancellano i dati non utilizzati ma ancora presenti in RAM. Se il rischio da cui vogliamo proteggerci è il furto dell'hardware il problema principale non è la RAM ma sono i dischi. Purtroppo il problema è più difficile per i dati presenti sul disco, sia il documento stesso nella sua cartella originale che nell'area di swap ove potrebbe esserci un'altra copia. Il programma di sistema di *remove* non cancella per nulla il documento dal disco, ne toglie solamente l'indirizzo dalla directory, e sarebbe quindi più giusto chiamarlo *unlink*. Esistono dei programmi che cancellano effettivamente il documento dal disco riscrivendoci sopra molte volte dati casuali e poi azzerando esattamente tutta l'area del disco. E' possibile fare ciò sia per l'area di disco ove vi sono documenti in stato *unlink* che per lo swap (previo averlo smontato!). Va notato però che questi metodi non danno la garanzia assoluta che il disco non si ricordi di cosa c'era precedentemente. Ci sono però due problemi che possono vanificare questi sforzi. Il primo è il fatto che i moderni dischi hanno delle cache incorporate (ad esempio con hardware cache-controller), le quali potrebbero rimandare di scrivere i dati su disco fino a che non vi sia la versione finale, perciò cancellando del tutto l'effetto della riscrittura dei dati. Il secondo è che molti filesystem oggi fanno *journalling* ovvero mantengono copia degli ultimi dati scritti su disco per permettere una veloce ricostruzione dei dischi in caso di crash del sistema. Non è chiaro come il journalling interferisca con la rimozione sicura di file e quanto (in linea di principio **tutto**!) sia effettivamente possibile ricostruire partendo dai dati di journalling. Vi sono poi delle situazioni particolari ancora più delicate, come ad esempio il caso in cui abbiamo rimosso con un unlink un file, poi il sistema ha allocato parte dello spazio usato dal vecchio file ad un nuovo file prima che noi facessimo la vera rimozione con riscrittura. Anche se ora lo spazio disco è usato da un altro file, è possibile ricostruire quello che vi stava precedentemente. Come possiamo procedere ora? Una possibile soluzione è abbandonare GPG ed usare un filesystem cifrato invece di cifrare i singoli file. In questo modo i documenti non esistono mai su disco in versione non cifrata, ad eccezione possibilmente dello swap che possiamo però cifrare anch'esso. Ci occuperemo di

3 Su alcune piattaforme `gpg` si lamenta che sta usando *insecure memory*, questo vuol dire che non è in grado di ottenere un *lock* su di una area di memoria RAM e che quindi è tecnicamente possibile che qualcuno (l'amministratore ad esempio) possa copiare i dati privati, come le chiavi segrete, che sono in elaborazione.

filesystem cifrati ed ancora altre accortezze per rendere più sicuri i documenti in un futuro articolo in questa rubrica.

Andrea Pasquinucci
Consulente di Sicurezza Informatica
pasquinucci@ucci.it

Riferimenti Bibliografici:

[1] OpenPGP Alliance, http://www.opengpg.org/

[2] RFC2440, J. Callas, L. Donnerhacke, H. Finney, R. Thayer, *OpenPGP Message Format* [November 1998]

[3] GnuPG, http://www.gnupg.org/

[4] PGP Coprporation, http://www.pgp.com/

[5] S. Garfinkel, *Pretty Good Privacy*, O'Reilly

[6] Peter Gutmann, *Secure Deletion of Data from Magnetic and Solid-State Memory*, 6th Usenix Security Symposium, 1996,
http://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html

[7] DoD 5220.22-M, http://www.dss.mil/isec/nispom.htm,
http://www.zdelete.com/dod.htm

[8] Secure Delete
http://www.thehackerschoice.com/download.php?t=r&d=secure_delete-2.3.tar.gz,
Wipe http://wipe.sourceforge.net/

```
> gpg --version
gpg (GnuPG) 1.0.6
Copyright (C) 2001 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Home: ~/.gnupg
Supported algorithms:
Cipher: 3DES, CAST5, BLOWFISH, RIJNDAEL, RIJNDAEL192, RIJNDAEL256, TWOFISH
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA, ELG
Hash: MD5, SHA1, RIPEMD160
```

Tabella 1

```
> gpg --homedir /mnt/floppy --no-options --lock-never --gen-key
gpg (GnuPG) 1.0.6; Copyright (C) 2001 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
Your selection? 1
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024) 2048
Do you really need such a large keysize? y
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct (y/n)? y

You need a User-ID to identify your key; the software constructs the user id
from Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"
Real name: Mister A
Email address: misterA@email.it
Comment: Dr.
You selected this USER-ID:
    "Mister A (Dr.) <misterA@email.it>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o

You need a Passphrase to protect your secret key.
Enter passphrase:
Repeat passphrase:

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
++++++++++++++++++++++++++++++++++++++++++++++++++ [ snip ]
public and secret key created and signed.
```

Tabella 2