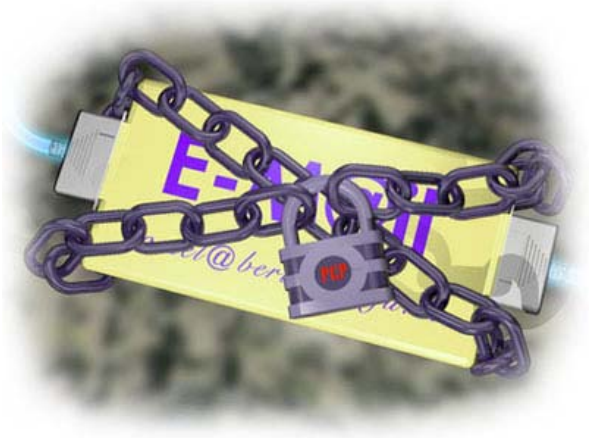
Andrea Grandi

# PGP. Mettere i propri dati al sicuro

marzo 2002



APOGEOnline
http://www.apogeonline.com

APOGEO

PGP. Mettere i propri dati al sicuro

Autore: Andrea Grandi



Viale Papiniano 38 – 20123 Milano (Italy)

Telefono: 02-461920 (5 linee r.a.) – Telefax: 02-4815382

Email apogeo@apogeonline.com

U.R.L. http://www.apogeonline.com

# Sicurezza e privacy in azienda


| | |
|---|---|
| **Autori**: | Paolo Galieri, Corrado Giustozzi, Marco Strano |
| **ISBN**: | 88-7303-970-7 |
| **Pagine**: | 142 |
| **Formato**: | 13,5 x 21 |
| **Disponibilità**: | Si |
| **Prezzo**: | € 12,40 |
| **Data di uscita**: | Dicembre 2001 |
| **Collana**: | Connessioni |

Un libro in tre parti: una tecnologica, una psicologica e una legale. La prima è un'introduzione divulgativa alle problematiche di sicurezza e privacy connesse all'uso crescente di "interazione digitale". La seconda è frutto dell'esperienza dell'autore nell'attività di prevenzione del crimine aziendale. La terza analizza le problematiche giuridiche del computer crime aziendale con un approccio chiaro e pratico.

## Sommario

# PGP: mettere i propri dati al sicuro

PGP (Pretty Good Privacy) è uno tra i più noti programmi che vengono utilizzati per proteggere dati riservati tramite l'utilizzo di tecniche di *crittografia*. Questo programma non si basa su un algoritmo in particolare, ma è in grado di gestire quasi tutti quelli più diffusi: RSA, IDEA, DSS (compatibile anche con GnuPG). Quando abbiamo la necessità di fare in modo che i nostri dati siano al sicuro e che possano venir letti solo da chi vogliamo noi, ricorriamo all'uso di questi algoritmi. Andiamo a vedere in dettaglio di cosa si tratta.

## 1. Cos'è la crittografia

Crittografare significa codificare un documento in modo che possa essere decifrato solo da chi ne conosce il "segreto". Per fare un esempio classico, è un po' come quando da bambini ci divertivamo ad inventare un alfabeto nostro in modo da potersi scambiare messaggi segreti.

Immaginiamo di voler mandare ad un nostro amico il messaggio "Ciao Giovanni" senza che gli altri possano leggerlo. Io e Giovanni potremmo concordare in precedenza di codificare il messaggio sostituendo ogni lettera con la successiva nell'alfabeto. Il messaggio diventerebbe così: "Dlbp Hlpzbool" che ai più risulterà illeggibile o sensa alcun senso. Il nostro amico Giovanni però sapendo il metodo utilizzato, non dovrebbe far altro che sotituire ogni carattere con quello che precede e tornare di nuovo all'originale "Ciao Giovanni". Il metodo qui descritto è abbastanza elementare e particolarmente insicuro, ma serve a spiegare cosa vuol dire cifrare un messaggio.

I moderni metodi di crittografia sono a chiave *simmetrica* (dove la chiave è segreta) e *asimmetrica* (dove la chiave è invece pubblica).

I principali algoritmi a chiave privata sono: DES, IDEA e RC4. Un algoritmo a chiave privata ha il vantaggio che le chiavi usate sono molto corte, quindi facili da ricordare e che le operazioni codifica/decodifica possono essere effettuate anche da computer poco potenti. Lo svantaggio maggiore è che bisogna utilizzare una chiave per ogni persona con la quale si vuole comunicare, inoltre i due utenti dovrebbero scambiarsi le chiavi di persona, per evitare che vengano intercettate.

Gli algoritmi a chiave pubblica più diffusi sono: RSA e DSS. Questi due algoritmi hanno l'enorme vantaggio di essere molto sicuri. Non dobbiamo preoccuparci infatti di trasmettere segretamente la chiave di crittazione, anzi potremmo inviarla in chiaro tramite un e-mail. L'unico svantaggio è dato dal fatto che questi algoritmi sono molto lenti e richiedono processori abbastanza potenti per svolgere le operazioni di codifica/decodifica. Quest'ultimo metodo è quello utilizzato dal PGP.

Ogni utente dispone infatti di una coppia di chiavi, una pubblica ed una privata. Se un nostro amico volesse inviarci informazioni riservate dovrebbe utilizzare la nostra chiave pubblica, e soltanto noi con la relativa chiave privata saremmo in grado di decifrare il messaggio. L'unico metodo possibile per decifrare il messaggio senza la chiave privata sarebbe quello di usare il brute force. In pratica dovremmo far provare al nostro computer tutte le chiavi possibili. Il fatto è che anche uno dei più moderni computer impiegherebbe centinaia di anni per trovare una chiave a 1024

bit, immaginiamoci poi quanto impiegherebbe per trovare una chiave a 2048 (PGP permette di creare chiavi anche fino a 4096 bit). Gli algoritmi attuali sono comunque talmente potenti che l governo americano ne ha addirittura vietato l'esportazione per evitare che vengano utilizzati per scopi terroristici.

## 2. I principali algoritmi:

Ecco una breve panoramica sui principali algoritmi utilizzati nei programmi di crittografia.

### Algoritmi a chiave privata

**DES**: sta per Data Encryption Standard. Questo algoritmo nasce attorno agli anni '70 e fin dall'inizio venne largamente usato dal governo americano. Il DES è in grado di operare su blocchi di dati da 64 bit mediante chiavi lunghe 56 bit. Il suo funzionamento non è molto complicato, sfrutta infatti semplici trasposizioni od XOR per cifrare i dati. Viste le ridotte dimensioni della chiave, al giorno d'oggi questo sistema non è più efficiente. Basterebbe un computer di fascia media per decifrare in poco tempo un documento provando tutte le combinazioni tra i 56 bit. Una variante che nacque in seguito fu il 3DES (triple DES) che prevedeva l'impiego ripetuto per tre volte di seguito di questo algoritmo.

**IDEA**: International Data Encryption Algoritm, nato nel 1991. Molto simile al DES, anche l'IDEA opera su blocchi di dati di 64 bit, utilizzando però chiavi a 128 bit, che rendono questo metodo molto più sicuro del precedente, anche se si basa su semplici trasposizioni o XOR come il precedente. l'IDEA è coperto da brevetto e quindi non è possibile utilizzarlo liberamente (non è ad esempio presente in GnuPG).

**RC4**: Questo algoritmo fu sviluppato da Ron Rivest per conto della RSA ed ha il vantaggio di essere veloce e poter utilizzare chiavi di differente lunghezza. Inizialmente rimase segreto per circa 7 anni, fino a che un anonimo non pubblico il suo codice sorgente su un newsgroup. Questo sistema effettua tre crittazioni consecutive come il 3DES e risulta dalle due alle tre volte più veloce di esso. Il suo principale impiego è quello nelle applicazioni che implementano la sicurezza nelle connessioni web.

### Algoritmi a chiave pubblica

**DSS**: Basato sull'algoritmo Diffie-Hellman. La sua sicurezza si basa sulla difficoltà del calcolo dei logaritmi ed è sicuro quanto più la sua chiave è grande. Dal momento in cui è scaduto il brevetto, viene comunemente utilizzato all'interno di PGP.

**RSA**: sta per Rivest Shamir Adelman. E' senza dubbio l'algoritmo più utilizzato e la sua sicurezza si basa sulla difficoltà di fattorizzazione dei grandi numeri, inoltre può utilizzare chiavi molto lunghe (fino a 2048 bit). Il suo brevetto è scaduto a Settembre 2000 e da allora viene impiegato liberamente nel PGP.

## 3. Firme digitali

Quando riceviamo un messaggio (cifrato o in chiaro che sia) può sorgere il problema di capire se chi lo ha inviato è veramente il mittente che appare nel campo "From: ". In questo caso ci vengono in aiuto le firme digitali. Firmare un messaggio vuol dire porre in fondo ad esso una breve chiave (l'hash) composta usando il testo del messaggio e la chiave pubblica. Conoscendo la chiave pubblica del reale mittente saremmo in grado di capire se si tratta effettivamente di lui o

no. Inoltre avremmo la certezza che durante il tragitto il messaggio non sia stato modificato.

## 4. GnuPG: la soluzione open per PGP

GnuPG è la versione *open* basata sul programma originale PGP di **Phiiip Zimmerman**. Questo programma non contiene l'algoritmo **IDEA** e quindi non è soggetto a restrizioni. GnuPG è disponibile all'indirizzo **http://www.gnupg.org** e al momento della scrittura di questo articolo l'ultima versione disponibile è la 1.0.6, che per altro corregge alcune vulnerabilità delle versioni precedenti. Per installare una copia di gnupg è possibile affidarsi ai comodi RPM (se supportati dal nostro sistema) ai file DEB (se utilizziamo Debian) oppure usare i più classici tar.gz; la procedura di installazione si riferirà in particolare a quest'ultimo metodo.

- Scompattare i sorgenti con il comando: **tar xfvz gnupg-x.y.z.tar.gz**
- entrare nella directory dei sorgenti e scrivere: **./configure** (si consiglia un ./configure --help per vedere tutte le opzioni disponibili).
- A questo punto basta compilare con il comando: **make**
- ed infine, se non ci sono stati errori durante la compilazione, digitare: **make install**

Il programma è installato nel nostro sistema. Dobbiamo provvedere adesso a generare la nostra coppia di chiavi. Per farlo basta fare: **gpg --gen-key** e ci apparirà:

```
gpg (GnuPG) 1.0.6; Copyright (C) 2001 Free
Software Foundation, Inc.
```

```
This program comes with ABSOLUTELY NO WAR-
RANTY.
This is free software, and you are welcome to
redistribute it
under certain conditions. See the file COPY-
ING 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?
```

Si consiglia la scelta numero 1, in seguito ci verrà chiesto di indicare la lunghezza della chiave:

```
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)
```

una chiave di 2048 bit sarà più che sufficiente per proteggere i nostri dati.

In seguito ci verrà chiesto quando scadrà la nostra chiave. Possiamo infatti fare in modo che la nostra chiave non sia più valida dopo una certa data:

```
Please specify how long the key should be va-
lid.
```

```
         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)
```

Successivamente ci verrà chiesto di indicare il nostro Nome, Email e un commento. che andranno a formare l'identificativo all'interno del portachiavi:

```
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) <hein-
richh@duesseldorf.de>"


Real name: Pippo
Email address: pippo@prova.com
Comment: pippo
You selected this USER-ID:
    "Pippo (pippo) <pippo@prova.com>"
Change (N)ame, (C)omment, (E)mail or
(O)kay/(Q)uit? o
```

Ora ci verrà chiesto di specificare una *passphrase*, ovvero una parola chiave che ci verrà chiesta ogni volta che vogliamo decifrare un nostro messaggio.

```
You need a Passphrase to protect your secret
key.

Enter passphrase: ************
```

A questo punto verranno generate le due chiavi (occorrerà circa 1 minuto a seconda della potenza del proprio computer, e nel frattempo verrà fatto un utilizzo intensivo della CPU).

Adesso il programma è pronto per essere utilizzato. Indicherò brevemente i comandi principali, mentre per gli altri rimando al comodissimo manuale (basta sigitare **man gpg** :p).

Esportare la nostra chiave in un file di testo: **gpg -a --export -o nomefile.asc**

In questo modo la nostra chiave può essere messa all'interno della nostra pagina web oppure allegata come attacchment ad ogni messaggio di posta elettronica che inviamo. Saremo così sicuri che tutti i nostri corrispondenti abbiano una copia della nostra chiave.

Importare una chiave nel portachiavi: **gpg --import [nome_del_file]**

PGP si basa infatti su un portachiavi. Si tratta di un piccolo database dove sono contenute le informazioni (l'ID) e le chiavi pubbliche vere e proprie dei nostri corrispondenti. Per cifrare un messaggio basterà fare riferimento all'ID del destinatario, verrà così sceltya la chiave appropriata.

Visualizzare le chiavi presenti nel database: **gpg --list-keys**

Con questo comando potremo veder ela lista delle chiavi presenti nel nostro database. Oltre all'ID dell'utente, potremo vedere anche la data in cui la chiave è stata creata.

Per cifrare un file: gpg -e [id_del_ricevente] [nome_del_file]

In questo modo codifichiamo un file e questo potrà essere decifrato soltanto dal destinatario che abbiamo scelto. Sarebbe opportuno cifrare il file utilizzando anche la nostra chiave, altrimenti nemmeno noi saremmo in grando di risalire al contenuto del file.

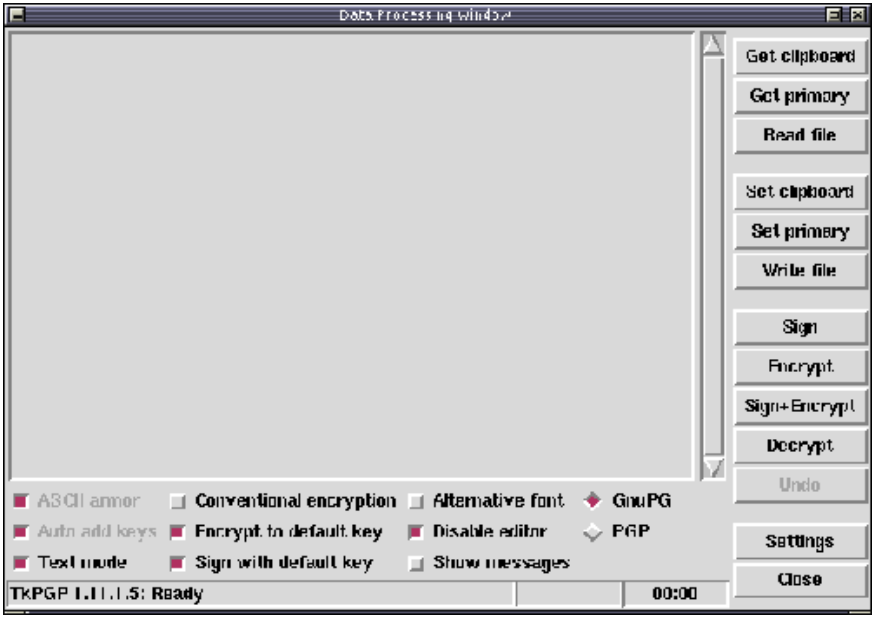
Per decifrare un messaggio: **gpg -d [nome_del_file]**

GPG questa volta utilizzerà la nostra chiave segreta per poter decifrare il messaggio. Ricordiamo ancora una volta che è fondamentale non dare MAI a nessuno la nostra chiave privata e soprattutto di non utilizzare GPG in remoto durante una sessione telnet. Il telnet infatti fa viaggiare i dati in chiaro e la nostra chiave potrebbe venir intercettata. Si consiglia in questo caso di utilizzare SSH.

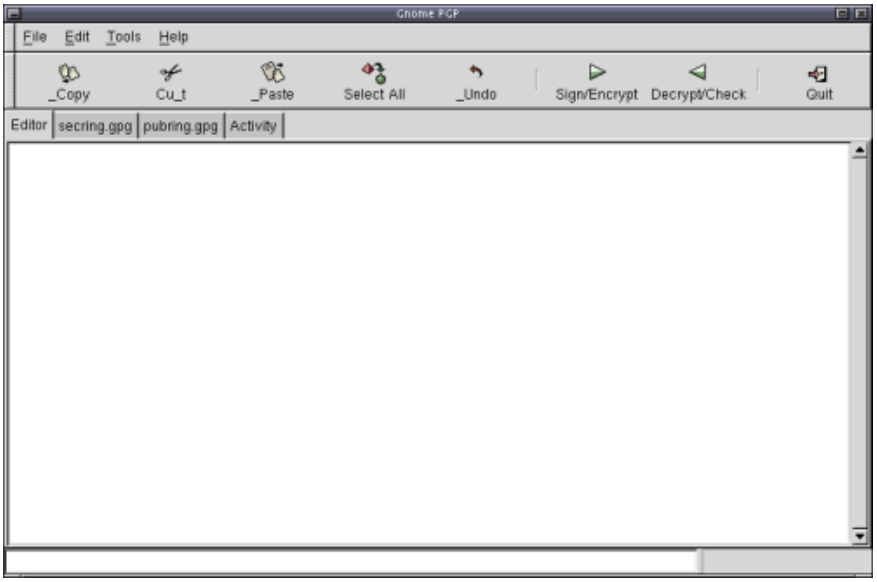
## 5. Frontend e integrazione con altri software

Esistono comodi tool che potete trovare sul sito ufficiale del GnuPG che facilitano l'utilizzo di questo programma. Basta andare nella sezione frontend e sce-

gliere quello che più ci piace. A mio avviso vorrei segnalare TkPGP che trovo ottimo per la sua velocità, leggerezza e soprattutto portabilità (è scritto in Tcl/Tk):

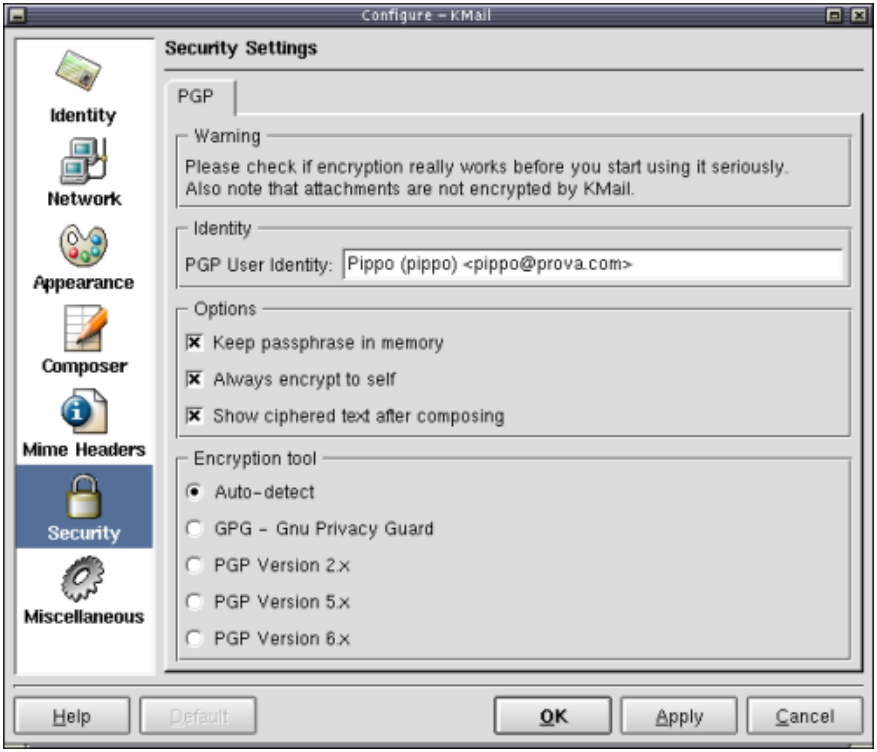


oppure GnomePGP:




Tra i client di posta che supportano PGP consiglio: **Kmail** ed **Evolution**. La configurazione di questi due client è veramente facile; basta indicare nell'apposta sezione l'ID della nostra chiave (nel caso del nostro esempio: "Pippo (pippo) <pippo@prova.com>".

In Kmail:


Configure – KMail
Identity
Network
Appearance
Composer
Mime Headers
Security
Miscellaneous
Security Settings
PGP
Warning
Please check if encryption really works before you start using it seriously.
Also note that attachments are not encrypted by KMail.
Identity
PGP User Identity:
Pippo (pippo) <pippo@prova.com>
Options
Keep passphrase in memory
Always encrypt to self
Show ciphered text after composing
Encryption tool
Auto-detect
GPG - Gnu Privacy Guard
PGP Version 2.x
PGP Version 5.x
PGP Version 6.x
Help
Default
OK
Apply
Cancel

ed in Evolution:


Siamo giunti alla conclusione di questa panoramica sugli strumenti di crittografia disponibili per i sistemi Unix/Linux. Ovviamente ne esistono molti altri, ma ho voluto inserire quelli che mi sembravano più efficienti e user-friendly.

Documento a cura di
***Andrea Grandi***