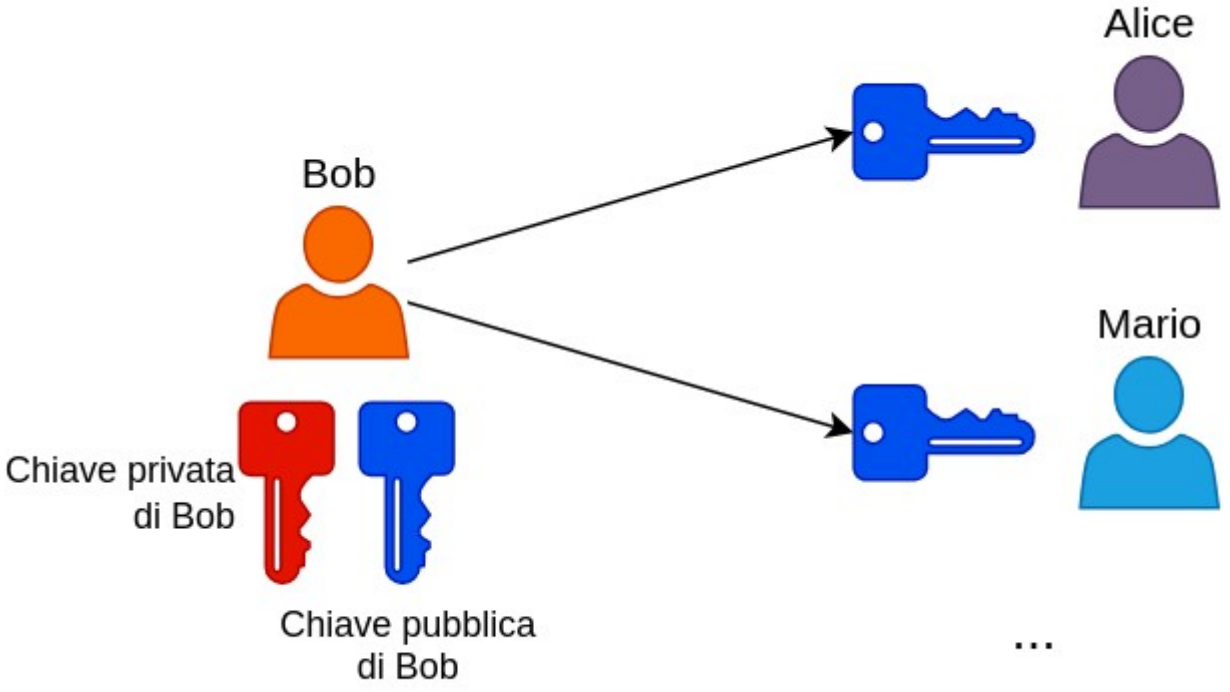
Dai comandi base all'uso con smartcard

Sonia Zorba

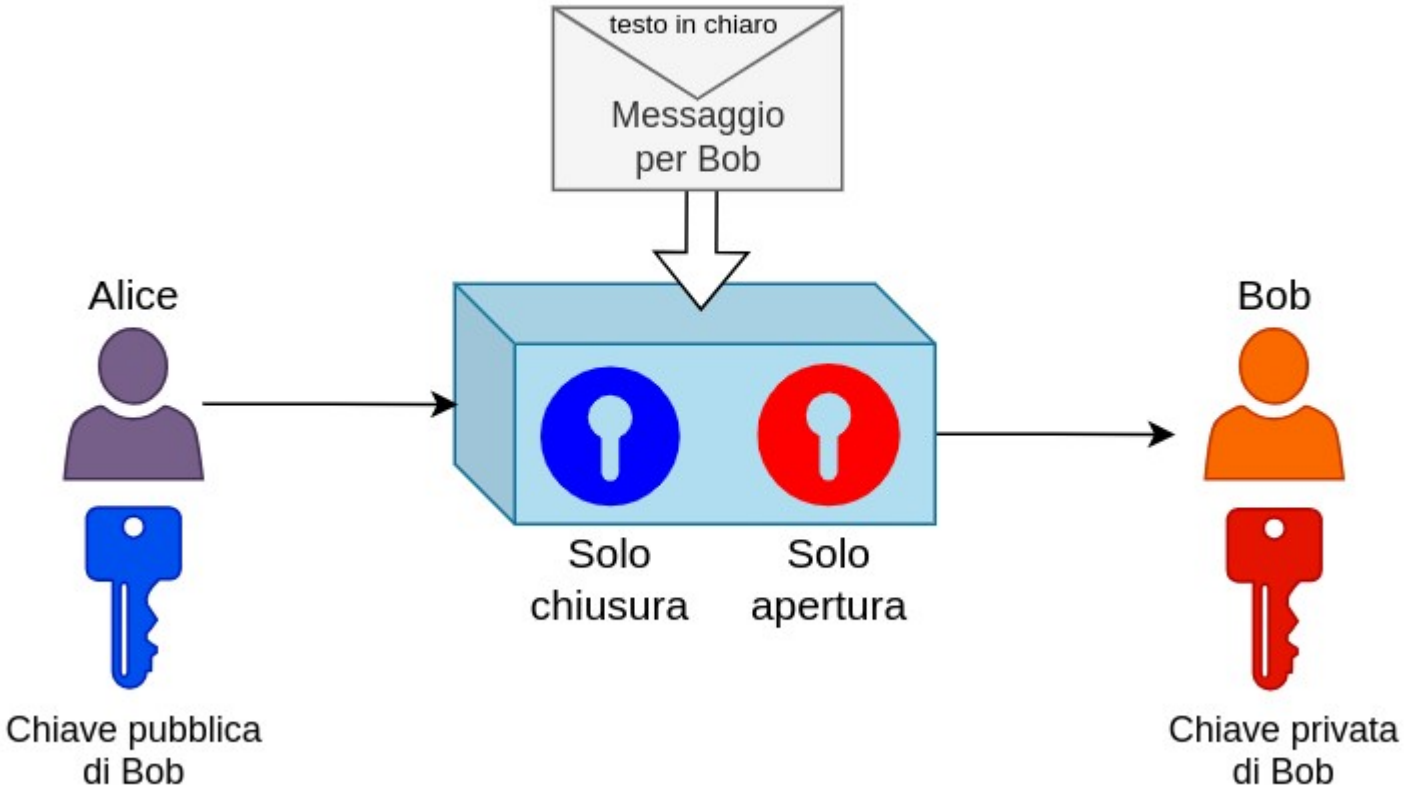
Lug Trieste

Linux Day 2024

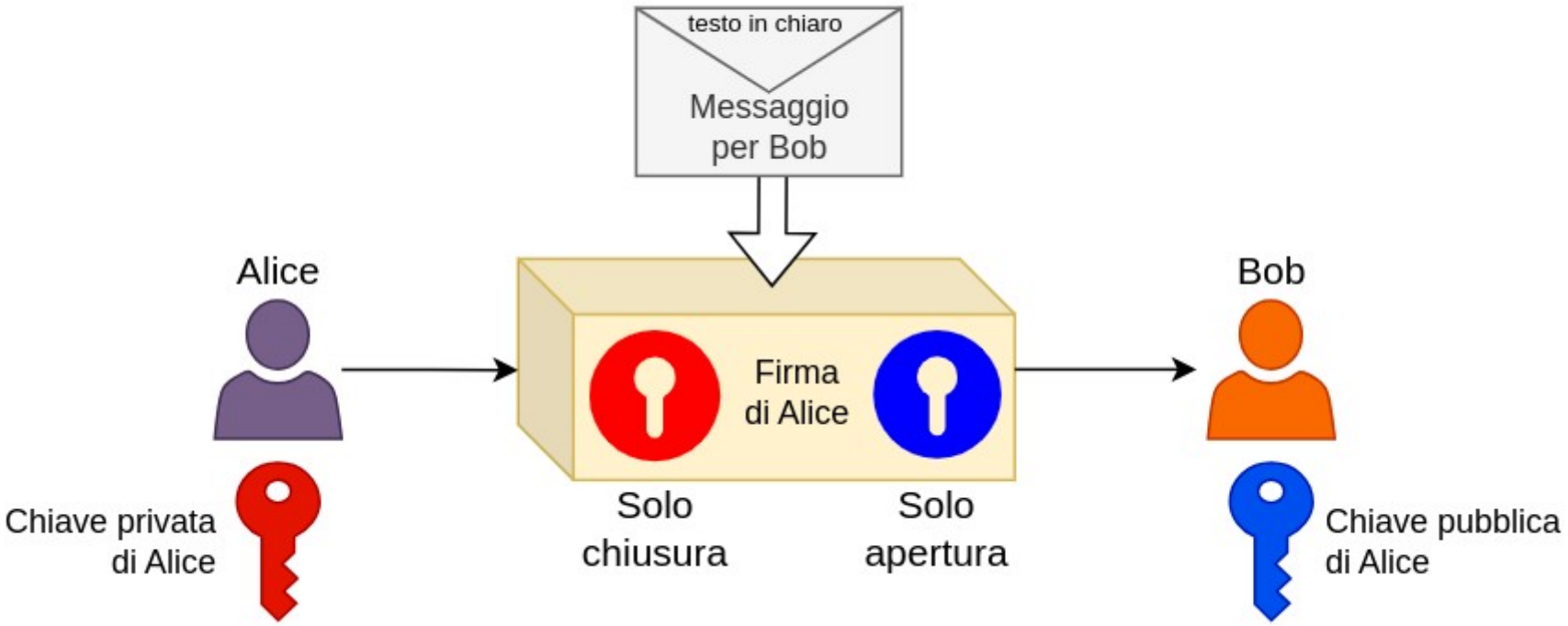
Crittografia asimmetrica



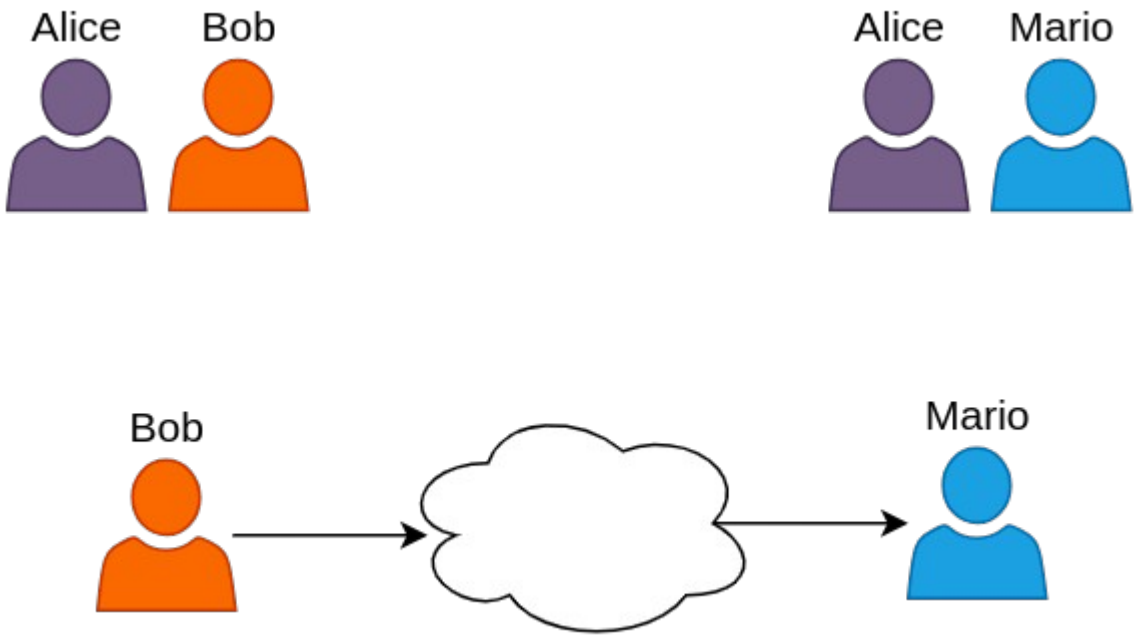
Cifratura asimmetrica



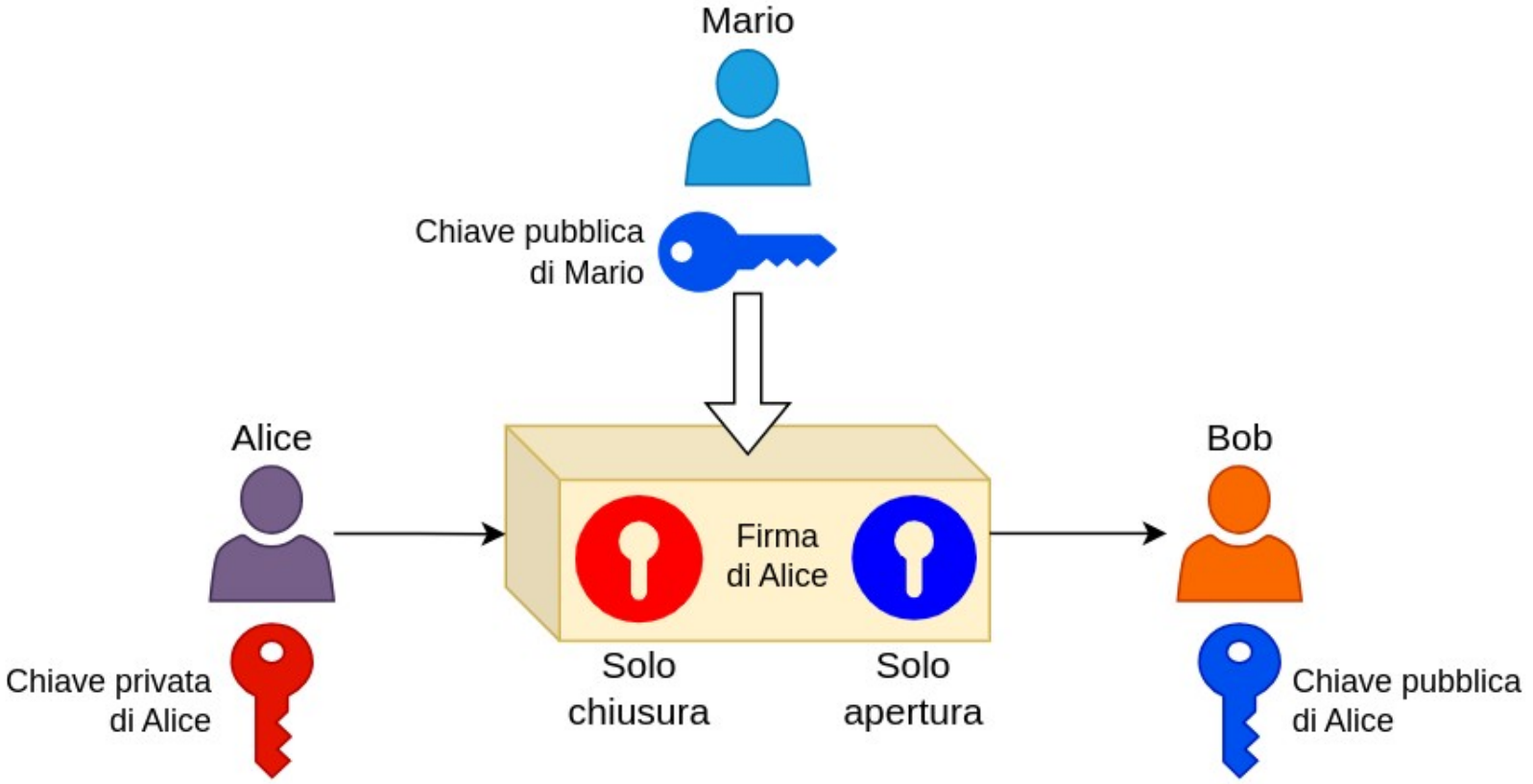
Firma digitale



Rete di fiducia



Firma di chiavi di altre persone



Usi

- Inviare e ricevere messaggi cifrati (soprattutto mail)
- Verificare che i pacchetti che scaricate siano stati realmente prodotti dai maintainer ufficiali
- Firmare i commit su git
- Cifrare password e file personali
- Sbloccare cifratura del disco (combinato con LUKS)
- Integrazione con SSH
- …

Un po’ di nomi…

1991

PGP

Implementazione

Licenza proprietaria

Pretty Good Privacy

→

1998

OpenPGP

Standard

→

1999

GPG

Implementazione

GPLv3

GNU Privacy Guard

Un po’ di nomi…

1991

PGP

Implementazione

Licenza proprietaria

Pretty Good Privacy

→

1998

OpenPGP

Standard

↓

LibrePGP

Standard

2023

?

↗

1999

GPG

Implementazione

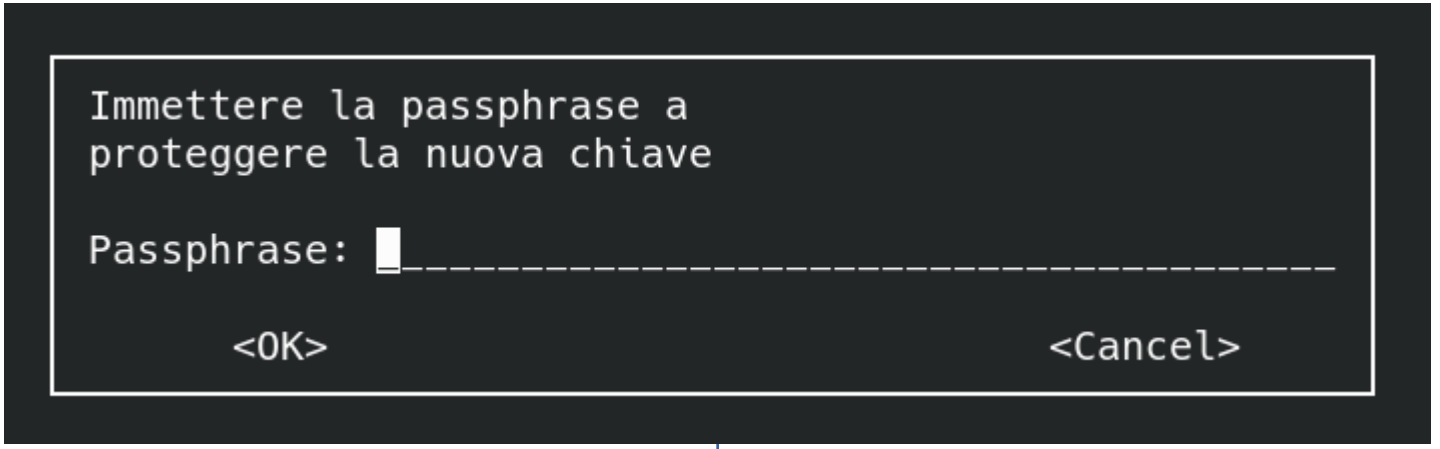
GPLv3

GNU Privacy Guard

Generare coppie di chiavi

```
$ gpg --generate-key

gpg: directory '/home/mario/.gnupg' creata
gpg: keybox '/home/mario/.gnupg/pubring.kbx' creato
Note: Use "gpg --full-generate-key" for a full featured key
generation dialog.

GnuPG deve costruire un ID utente per identificare la chiave.

Nome e Cognome: Mario Rossi
Indirizzo di Email: mario@example.com
Hai selezionato questo User Id:
    "Mario Rossi <mario@example.com>"

Modifica (N)ome, (C)ommento, (E)mail oppure (O)kay/(Q)uit? O
```

Generare coppie di chiavi



Questa UI dipende dal tool pinentry

- pinentry-gnome3
- pinentry-curses
- pinentry-x11

Generare coppie di chiavi

Dobbiamo generare un mucchio di byte casuali. È una buona idea eseguire qualche altra azione (scrivere sulla tastiera, muovere il mouse, usare i dischi) durante la generazione dei numeri primi; questo dà al generatore di numeri casuali migliori possibilità di raccogliere abbastanza entropia.

Generare coppie di chiavi

```
gpg: /home/mario/.gnupg/trustdb.gpg: creato il trustdb
gpg: directory '/home/mario/.gnupg/openpgp-revocs.d' creata
gpg: certificato di revoca archiviato come
'/home/mario/.gnupg/openpgp-revocs.d/355C0EBBDC8479744B5BFE98CC95D3
6BDC5A40DD.rev'
chiavi pubbliche e segrete create e firmate.

pub   ed25519 2024-09-30 [SC] [expires: 2027-09-30]
      355C0EBBDC8479744B5BFE98CC95D36BDC5A40DD
uid                      Mario Rossi <mario@example.com>
sub   cv25519 2024-09-30 [E] [expires: 2027-09-30]
```

I file in ~/.gnupg

Attenzione alle versioni!

Legacy	Nuove (GPG 2.1+)
pubring.gpg secring.gpg trustdb.gpg	pubring.kbx private-keys-v1.d/*.key trustdb.gpg

Chiavi subordinate (subkeys)

```
pub   ed25519 2024-09-30 [SC] [expires: 2027-09-30]
      355C0EBBDC8479744B5BFE98CC95D36BDC5A40DD
uid                      Mario Rossi <mario@example.com>
sub   cv25519 2024-09-30 [E] [expires: 2027-09-30]
```

- Le chiavi hanno varie “capabilities”.
- Non c’è differenza tecnica, cambia l’uso che se ne fa.
- La chiave **master**:
  - è quella abilitata a firmare altre chiavi (“C”: certify);
  - rappresenta l’identità dell’utente;
  - può essere usata generare chiavi subordinate.

Chiavi subordinate (subkeys)

Perché usarle?

- master key compromessa == molto male
- subkey compromessa == male, ma non malissimo
  - Posso revocarla separatamente, senza compromettere la rete di fiducia costruita con la chiave primaria

Computer diversi → Diverse subkey
Hardware lento → Subkey più corta

Che tipo di chiave vuoi?

```
$ gpg --full-generate-key

Per favore scegli che tipo di chiave vuoi:
   (1) RSA and RSA
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
   (9) ECC (sign and encrypt) *default*
  (10) ECC (sign only)

Cosa scegli?
```

Default di GPG nel corso degli anni: DSA → RSA → ECC

RSA vs ECC

- RSA:
  - Più la chiave è lunga e più è sicura
  - Forte dipendenza dal generatore di numeri casuali usato: dei ricercatori sono riusciti a craccare 2 chiavi su 1000
- ECC:
  - La sicurezza dipende dalla curva scelta
    - Curve25519 , NIST P-256 (https://safecurves.cr.yp.to/)
  - Più veloce rispetto ad RSA
  - In alcune distro disponibile solo in expert mode (es: Debian 12)

Backup delle chiavi

Master key:

```
$ gpg --output backupkeys.pgp --armor --export-secret-keys \
    --export-options export-backup mario@example.com
```

```
$ gpg --export-secret-keys mario@example.com \
    | paperkey -o /tmp/key-backup.txt
```

Il contenuto dei file è comunque protetto da password!

Subkeys:

```
$ gpg --export-secret-subkeys mario@example.com
```

Export della chiave pubblica

```
$ gpg --export --armor mario@example.com

-----BEGIN PGP PUBLIC KEY BLOCK-----

mDMEZvurURYJKwYBBAHaRw8BAQdA6tiMUT49ZZ8KTgR6jQdwTrJz7C/zRgzT9HK6
YSt+q9K0H01hcmlvIFJvc3NpIDxtYXJpb0BleGFtcGxlLmNvbT6ImQQTFgoAQRYh
BJLLcFomCCFHO099NTLRkWg8WzRtBQJm+6tRAhsDBQkFo5qABQsJCAcCAiICBhUK
CQgLAgQWAgMBAh4HAheAAAoJEDLRkWg8WzRt0HkA/jyoxXExfEXXwXg2UMO8k+uB
DhbDrS5pV2MToAQCPScmAQDSUXXXcrFQHnuWHT3Ovw+LYc1vvNaVfY+/T3iXo/Jj
Cbg4BGb7q1ESCisGAQQBl1UBBQEBB0BcPaNv12cLPUku8YT6J8VtK5kuQM+0+u5H
qRTV2fgJMgMBCAeIfgQYFgoAJhYhBJLLcFomCCFHO099NTLRkWg8WzRtBQJm+6tR
AhsMBQkFo5qAAAoJEDLRkWg8WzRthgQBAONEtm68TxjZmmG5o4Fc3/0u1acGVdDY
Bi7VbL1PwmXOAP9m1V1/p2FoVviU92zEIBwrJBKx29wd9QLI2ZIkKZ59AA==
=9Wdt
-----END PGP PUBLIC KEY BLOCK-----
```

Import di una chiave pubblica

```
$ gpg --import alice.gpg
gpg: key 43B378258D9991D9: public key "Alice <alice@example.com>"
imported
gpg: Total number processed: 1
gpg:               imported: 1

$ gpg --list-keys
pub   ed25519 2024-10-01 [SC] [expires: 2027-10-01]
      BBC7A017A416DEFB7D29078243B378258D9991D9
uid           [ unknown] Alice <alice@example.com>
sub   cv25519 2024-10-01 [E] [expires: 2027-10-01]
```

Trust

```
$ gpg --edit-key alice@example.com
gpg> trust

[ sconosciuto] (1). Alice <alice@example.com>

Si prega di decidere fino a che punto si considera attendibile
questo utente per verificare correttamente le chiavi di altri
utenti (guardando i passaporti, controllando le impronte digitali
da fonti diverse, ecc.)
```

Trust

```
  1 = Non so o non dirò
  2 = Non mi fido
  3 - Mi fido marginalmente
  4 - Mi fido completamente
  5 = Mi fido in ultima analisi
  m = torna al menu principale

Cosa hai deciso? 5
Vuoi davvero impostare questa chiave per la massima fiducia? (y/N)
y
```

Trust

```
$ gpg --list-key alice@example.com

gpg: controllo il trustdb

gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid: 2  signed: 0   trust: 0-, 0q, 0n, 0m, 0f, 2u

gpg: il prossimo controllo del trustdb sarà fatto il 2025-06-16

pub   ed25519 2024-10-01 [SC] [scadenza: 2027-10-01]
      BBC7A017A416DEFB7D29078243B378258D9991D9
uid           [ultimo] Alice <alice@example.com>
sub   cv25519 2024-10-01 [E] [scadenza: 2027-10-01]
```

Keyserver

```
$ gpg --send-key [--keyserver …] mario@example.com

$ gpg --search-key [--keyserver …] alice@example.com
$ gpg --recv-keys [--keyserver …] DBB802B258ACD84F
$ gpg --sign-key [--keyserver …] DBB802B258ACD84F
$ gpg --send-keys [--keyserver …] DBB802B258ACD84F
```

- Default: https://keys.openpgp.org
- Svantaggio: non posso eliminare le chiavi inserite

Per sviluppatori

- Piattaforme basate su Git espongono la chiave pubblica dell'utente (se configurata):
  - https://github.com/zonia3000.gpg
  - https://codeberg.org/zonia3000.gpg
  - https://git.mittelab.org/zonia3000.gpg

- Si possono anche firmare i commit:

Esempio 1: cifrare un file

Cifrare un file usando una chiave pubblica:

```
$ gpg --encrypt --recipient mario@example.com pippo.txt
```

Produce come output il file `pippo.txt.gpg`

Per decifrare (richiede la password della chiave privata):

```
$ gpg --decrypt pippo.txt.gpg
```

Esempio 2: verifica pacchetti

- I pacchetti ufficiali Debian sono firmati con delle chiavi GPG memorizzate in `/etc/apt/trusted.gpg.d/`
- Il package manager (apt) valida le firme dei pacchetti in fase di installazione
- Posso anche scaricare un pacchetto contenente le chiavi personali degli sviluppatori Debian:

```
$ sudo apt install debian-keyring
```

- Queste chiavi possono essere usate per validare chiavi di altre distro (es: Tails)

Esempio 3: password manager

Pass: https://www.passwordstore.org/

```
$ pass init DB891A14
```

```
$ pass insert gitlab
```

```
$ ls ~/.password-store/
gitlab.gpg
```

Smartcard (o chiavette) GPG

Security token: mini computer che contiene le chiavi private e può eseguire operazioni crittografiche.
È più sicuro rispetto ad avere la chiave privata salvata sul disco.

Dipendenza: **scdaemon**

YubiKey

Inseriamo la card

```
$ gpg --card-status
Reader ...........: 20A0:42B2:X:0
Application ID ...: D2760001240103040000F10AB5C1D0000
Application type .: OpenPGP
Version ..........: 3.4
[...]
Max. PIN lengths .: 127 127 127
PIN retry counter : 3 0 3
Signature counter : 0
KDF setting ......: off
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]
General key info..: [none]
```

Smartcard (o chiavette) GPG

Step 1: generare le chiavi

Possibilità:

1) Far generare le chiavi alla card (no backup!)
2) Generare le chiavi su un computer sicuro, fare un backup e importarle nella card

Generazione chiavi expert mode

La card ha 3 slot: [SC], [E], [A]

Generiamo la chiave **SC** in expert mode:

```
$ gpg --expert --full-gen-key

Per favore scegli che tipo di chiave vuoi:
[...]
  (11) ECC (imposta le tue capacità)

Azioni correnti consentite: Firma Certifica

   (S) Attivare o disattivare la funzionalità di firma
   (A) Attivare/disattivare la funzionalità di autenticazione
   (Q) Finito
```

addkey 1

Aggiungiamo la chiave subordinata **E** in expert mode:

```
$ gpg --expert --edit-key 5AF6F3CE

gpg> addkey

[...]
  (12) ECC (solo crittografia)

sec  ed25519/A6CB41A05AF6F3CE
     creato: 2024-10-03  scadenza: mai         utilizzo: SC
     attendibilità: definitivo    validità: definitivo
ssb  cv25519/F002200B3FD32D06
     creato: 2024-10-03  scadenza: mai         utilizzo: E
```

addkey 2

Aggiungiamo la chiave subordinata **A** in expert mode:

```
gpg> addkey

[...]
  (11) ECC (imposta le tue capacità)

Azioni correnti consentite: Firma

   (S) Attivare o disattivare la funzionalità di firma
   (A) Attivare/disattivare la funzionalità di autenticazione
   (Q) Finito

gpg> save
```

Chiavi generate

Verifichiamo il risultato:

```
$ gpg --list-key 5AF6F3CE

pub   ed25519 2024-10-03 [SC]
      3E8E0C61670C5997F062C614A6CB41A05AF6F3CE
uid           [ultimo] Mario Rossi <mario@example.com>
sub   ed25519 2024-10-03 [A]
sub   cv25519 2024-10-03 [E]
```

E **facciamo un backup**! Le chiavi private vengono cancellate quando vengono fatte migrare sulla card!

Interagire con la smartcard

Verifico status:

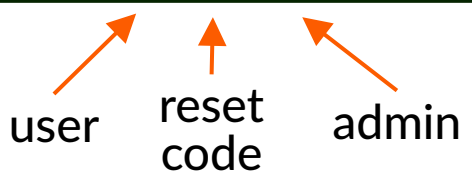
```
$ gpg --card-status
```

Modifica:

```
$ gpg --edit-card
gpg/card> help
quit              abbandona questo menù
admin             mostra comandi di amministrazione
help              mostra questo aiuto
list              elencare tutti i dati disponibili
[…]
```

Interagire con la smartcard

```
gpg/card> list
Serial number ....: XXXXXXXX
URL of public key : [non impostato]
Login data .......: [non impostato]
[…]
Signature key ....: [none]
Encryption key....: [none]
Authentication key: [none]
[…]
PIN retry counter : 3 0 3
```



Interagire con la smartcard

Cambiare i PIN:

```
gpg/card> passwd
1 - change PIN
2 - unblock PIN
3 - change Admin PIN
4 - set the Reset Code
Q - quit
```

Keytocard 0

Spostare la chiave primaria sulla card:

```
$ gpg --expert --edit-key 5AF6F3CE

gpg> keytocard
Spostare davvero la chiave primaria? (y/N) y
Si prega di selezionare dove memorizzare la chiave:
   (1) Chiave di firma
   (3) Chiave di autenticazione
Cosa scegli? 1
```

Abbiamo riempito il primo slot: [SC]

Keytocard 1

Selezionare la seconda chiave [E]:

```
gpg> key 1
sec  ed25519/FC1B4CEB7A62D033
     creato: 2024-10-04  scadenza: mai         utilizzo: SC
     scheda-no: 000X XXXXXXXX
     attendibilità: definitivo    validità: definitivo
ssb* cv25519/2B00EFC4A5B8579C
     creato: 2024-10-04  scadenza: mai         utilizzo: E
ssb  ed25519/EB6E036ABEC6F35F
     creato: 2024-10-04  scadenza: mai         utilizzo: A
```

Keytocard 1

Spostare la chiave [E] sulla card:

```
gpg> keytocard
Spostare davvero la chiave primaria? (y/N) y
Si prega di selezionare dove memorizzare la chiave:
   (2) Chiave di cifratura
   (3) Chiave di autenticazione
Cosa scegli? 2
```

Abbiamo riempito il secondo slot: [E]

Keytocard 2

Deselezionare la seconda chiave [E] e selezionare la terza [A]:

```
gpg> key 1
gpg> key 2
sec  ed25519/FC1B4CEB7A62D033
     creato: 2024-10-04  scadenza: mai         utilizzo: SC
     scheda-no: 000F 10AB5C1D
     attendibilità: definitivo    validità: definitivo
ssb  cv25519/2B00EFC4A5B8579C
     creato: 2024-10-04  scadenza: mai         utilizzo: E
     scheda-no: 000F 10AB5C1D
ssb* ed25519/EB6E036ABEC6F35F
     creato: 2024-10-04  scadenza: mai         utilizzo: A
```

Keytocard 2

Spostare la chiave [A] sulla card:

```
gpg> keytocard
Si prega di selezionare dove memorizzare la chiave:
   (3) Chiave di autenticazione
Cosa scegli? 3
```

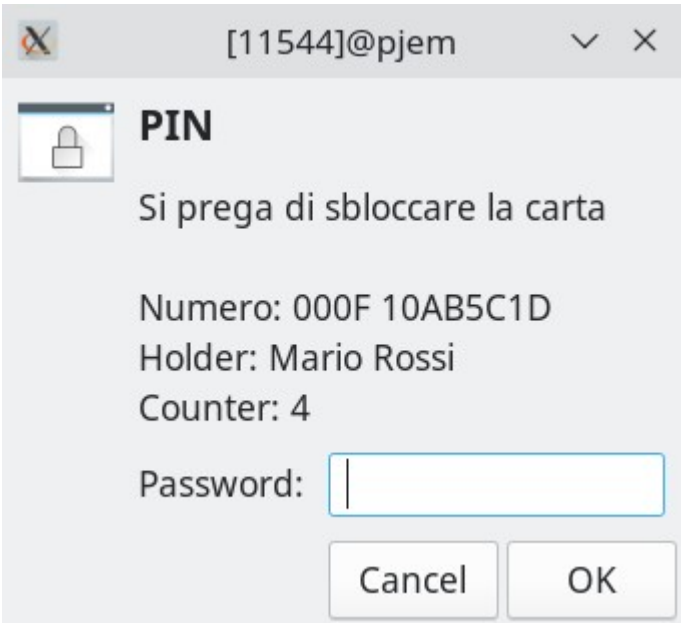
Abbiamo riempito anche il terzo slot: [A]

Per verificare il risultato:

```
$ gpg --card-status
```

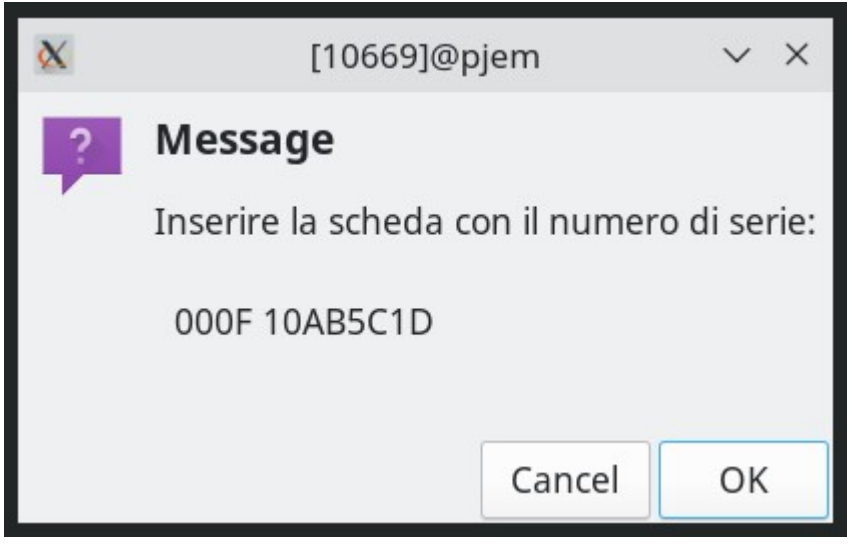
La carta è pronta!

```
$ gpg --sign pippo.txt
```



Stub

La chiave segreta non è più nel computer, al suo posto c'è uno "stub", una sorta di puntatore alla smartcard.



Importare uno stub?

Come posso usare la smartcard su un altro computer? Se la inserisco e provo ad usarla mi dà:

```
gpg: signing failed: No secret key
```

Basta importare la chiave pubblica affinché venga creato lo stub per la carta:

```
$ gpg --import mario-public-key.gpg
$ gpg --card-status
```

Non esiste un modo (semplice) di estrarre la chiave pubblica dalla smartcard! Meglio avere una copia.

Integrazione con LUKS

/etc/crypttab

Configurazione classica:

```
sda5_crypt UUID=0b78c1f7-a724-46ce-8174-d0396f02292e
none luks
```

Configurazione con GPG (Debian):

```
sda5_crypt UUID=c87bd175-aa32-437d-8bea-cddd2e4fc12c
/etc/cryptsetup-initramfs/cryptkey.gpg
luks,keyscript=decrypt_gnupg-sc
```

`/lib/cryptsetup/scripts/decrypt_gnupg-sc`

Integrazione con LUKS

Generazione della chiave:

```
$ dd if=/dev/urandom bs=1 count=256 > cryptkey
```

Cifratura con GPG:

```
$ gpg --recipient mario@example.com \
    --output /etc/cryptsetup-initramfs/cryptkey.gpg \
    --encrypt cryptkey
```

Integrazione con LUKS

Aggiunta della chiave a disco LUKS pre-esistente:

```
cryptsetup luksAddKey --new-keyfile=cryptkey /dev/sda5
```

Il comando ha aggiunto un nuovo keyslot:

```
# cryptsetup luksDump /dev/sda5

Keyslots:
  0: luks2  <-------- Vecchia passphrase (ancora valida)
      ...
  1: luks2  <-------- Chiave appena generata
      ...
```

Integrazione con LUKS

Aggiorniamo initramfs:

```
# update-initramfs -u
```

(un apposito hook interpreta /etc/crypttab e aggiunge il necessario)

Se usiamo l'opzione splash, aggiorniamo anche grub:

```
GRUB_CMDLINE_LINUX_DEFAULT="nosplash"
```

```
# update-grub
```

Reboot!

Riferimenti

- `man gpg`
- https://gnupg.org/
- https://github.com/drduh/YubiKey-Guide
- https://cryptsetup-team.pages.debian.net/cryptsetup/README.gnupg-sc.html