La crittografia al servizio di Linux, conservare in maniera sicura i propri dati.

Flavio Castelli aka micron <micron@madlab.it>

Disclaimer

Tutto il materiale contenuto in questa presentazione ha fini esclusivamente informativi ed educativi.

L'autore non si ritiene in alcun modo responsabile per danni perpetrati a cose o persone causati dall'uso di codice, programmi, informazioni, tecniche contenuti all'interno della stessa.

Introduzione: cosa è la crittografia?

Definizione

La crittografia è l'arte di progettare algoritmi (o cifrari) per crittografare un messaggio rendendolo incomprensibile a tutti tranne al suo destinatario.

Solo chi è in possesso di una determinata chiave è in grado di leggere il testo in chiaro.

Introduzione: cosa è la crittografia?

*Un mito da sfatare**

La ***robustezza*** di un sistema di crittografia risiede solo ed esclusivamente nella segretezza della chiave e non dell'algoritmo.
È opportuno rendere pubblico l'algoritmo, in modo che se ne possano scoprire eventuali punti deboli in anticipo.

La ***sicurezza*** del sistema dipende esclusivamente dalla segretezza della chiave.

* principio enunciato nel 1883 da Auguste Kerckhoffs

Introduzione: cosa è la crittografia?

Cenni storici (1)

- Antico egitto: primo uso della crittografia, uso di geroglifici non standard.
- Il cifrario di Cesare: nelle comunicazioni militari ogni lettera dell'alfabeto era sostituita con quella presente alla posizione *p+x*, dove *p* è la posizione alfabetica del carattere originale.

Esempio:

Testo in chiaro: BG LUG
Algoritmo: $A=p_B+k$ *(con k=3)*
Testo cifrato: *EL OAL*

Legenda:
A=nuovo carattere (after)
p_B=posizione carattere non codificato (before)
k= chiave (key)

Introduzione: cosa è la crittografia?

Cenni storici (2)

Nel 1466 Leon Battista Alberti pubblica il "*De cifris*".
Sono illustrati i principali metodi di cifratura noti all'epoca, è introdotto un nuovo metodo, creato dall'autore, che prevede la modifica periodica della chiave.

Esempio:

Testo in chiaro: BG LUG
Algoritmo: $A=p_B+k$ (k=3 per la prima parola, k=4 per la seconda parola)
Testo cifrato: *EM QDP*

Legenda:
A=nuovo carattere (after)
p_B=posizione carattere non codificato (before)
k= chiave (key)

Introduzione: cosa è la crittografia?

Cenni storici (3)

Un secolo più tardi Blaise de Vigenère, partendo dall'idea di Alberti e dalle sue successive elaborazioni, creò un sistema di codifica particolarmente efficace, noto con il nome di: “cifrario indecifrabile”.

Si usa una tabella contenente tutti gli alfabeti di cifratura (a 26 lettere) disponibili.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
..																									
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Funzionamento:

- Si sceglie una parola chiave.
- Si identificano le le righe della tabella che iniziano con le lettere della parola chiave, ottenendo così degli alfabeti.
- Si cicla tra questi alfabeti cifrando il testo.

NB: la stessa lettera può essere cifrata in maniera diversa

Introduzione: cosa è la crittografia?

Cenni storici (4)

- Negli anni successivi furono introdotti dispositivi di cifratura meccanici ed elettromeccanici quali *enigma*, un dispositivo usato dall'esercito tedesco durante la seconda guerra mondiale.

- Con l'introduzione dell'informatica iniziò la sviluppo di algoritmi indicati per essere eseguiti dal computer.
  -->È finalmente possibile avere delle chiavi di dimensioni notevoli, rendendo improponibile un attacco manuale.

Tipi di crittografia

Chiave simmetrica

Per cifrare e decifrare i dati è usata un'unica chiave

Svantaggio principale: nel caso in cui si debba distribuire del materiale confidenziale siamo costretti a diffondere la chiave.

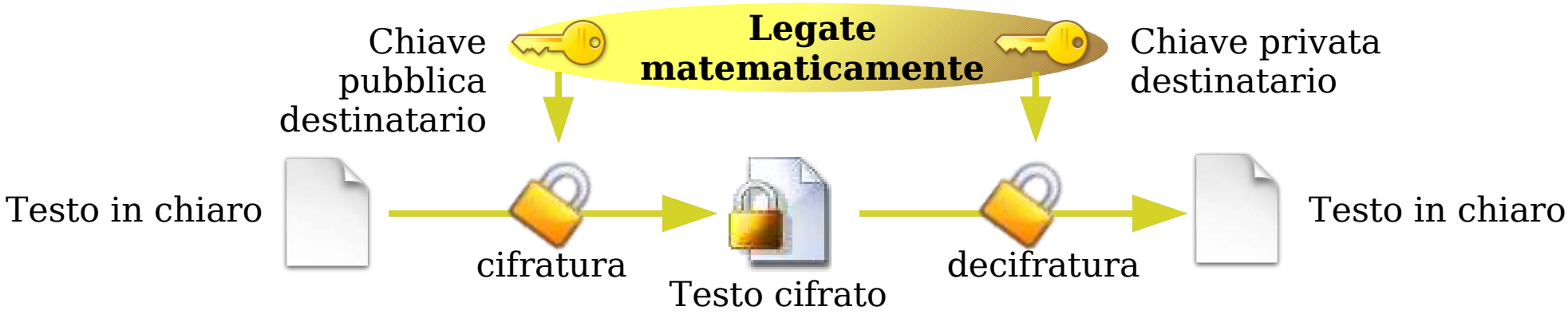
Come detto in precedenza la sicurezza di un sistema crittografico dipende esclusivamente dalla segretezza della chiave --> maggiore è il numero delle copie della chiave, maggiori sono le probabilità che i dati siano compromessi

Tipi di crittografia

Chiave asimmetrica - definizione (*)

Per cifrare e decifrare i dati sono usate due chiavi:

- Chiave pubblica: cifratura dati
- Chiave privata: decifratura dati



- Le due chiavi sono ricavate tramite un procedimento matematico, sono collegate tra di loro.
- È impossibile ricavare una chiave partendo dall'altra.
- La *chiave privata* non deve essere diffusa.
- La *chiave pubblica* è liberamente distribuibile.

* tecnica ideata nel 1975 da Whitfield Diffie e Martin Hellman

Crittografia in Linux

Cosa vogliamo fare?

Prima di tutto dobbiamo chiederci quali sono le nostre esigenze.

Possibili scenari d'uso:

- Inviare mail “sicure”
- Conservare in maniera sicura alcuni dati.
- Conservare in maniera sicura tutti i dati presenti su un device.
- Proteggere tutti i dati presenti nel sistema.

Invio mail sicure

Operazioni possibili

Possiamo compiere le seguenti azioni:

- **Cifrare la mail:** usiamo la chiave pubblica del destinatario, soltanto lui (usando la sua chiave privata) sarà in grado di leggerla
- **Firmare la mail:** usiamo la nostra chiave privata per firmare il messaggio, il destinatario (usando la nostra chiave pubblica) sarà in grado di stabilire l'autenticità del mittente.
- **Cifrare e firmare la mail:** firmiamo la mail usando la nostra chiave privata, mentre usiamo quella pubblica del destinatario per cifrare il messaggio. Si combinano gli aspetti elencati sopra.

Invio mail sicure

Strumenti a disposizione

Per inviare mail crittografate o firmate servono i seguenti programmi:

- **GPG (GNU Privacy Guard):** è lo strumento alla base di tutto, gestisce le chiavi, la cifratura e la decifratura.
- **Client di posta:** molti hanno supporto diretto a gpg (mutt, pine, kmail, sylpheed, evolution)
- **Fronted GPG:** evitano di dovere impare la sintassi di gpg, ne facilitano l'uso. Per Kde è presente "*kgpg*", per gli amanti delle gtk esiste "*GPA*".

Invio mail sicure

Uso di GPG – Creazione di una coppia di chiavi (1)

Creiamo una nuova coppia di chiavi per GPG:

```
micron@melindo micron $ gpg --gen-key
gpg (GnuPG) 1.2.4; Copyright (C) 2003 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.
```

Scelta dell'algoritmo:

```
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
```

Dimensioni delle chiavi:

```
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) 1024
Requested keysize is 1024 bits
```

Invio mail sicure

Uso di GPG – Creazione di una coppia di chiavi (2)

Periodo di validità:

```
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) 0
Key does not expire at all
Is this correct (y/n)? y
```

Identificativo:

```
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: Flavio Castelli
Email address: micron@madlab.it
Comment: Il mio indirizzo di posta principale
You selected this USER-ID:
    "Flavio Castelli (Il mio indirizzo di posta principale)
<micron@madlab.it>"
```

Conferma:

```
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.
```

Invio mail sicure

Uso di GPG – Comandi base

Elenco chiavi:

```
gpg --list-keys
```

Importare una chiave pubblica:

```
gpg –import [nome file]
```

Esportare una chiave in formato ascii:

```
gpg -a --export [UID]
```

Per indicare un UID relativo a una chiave si può usare l'indirizzo di posta del suo proprietario, oppure l'output fornito dal comando "*--list-keys*"

Revocare una chiave:

```
gpg --gen-revoke
```

Gestire una chiave:

```
gpg --edit-key UID
```

Consente di eseguire una serie di operazioni quali: firma della chiave, aggiunta di un indirizzo di posta secondario, cambio password, ...

Invio mail sicure

Uso di GPG – Cifrare e decifrare

- Cifrare un file:

```
gpg -e -r destinatario [file]
```

Otterremo così un file chiamato file.gpg contenente il messaggio crittografato.

- Decifrare un file:

```
gpg -d [file]
```

Il messaggio decifrato sarà stampato sullo stdout, per ovviare usare l'opzione "*-o*" seguita dal nome del file di destinazione (oppure si può ridirezionare l'output a mano).

Invio mail sicure

Uso di GPG – Firmare un file

- Firmare un file:

```
gpg -s [--armor] [file]
```

  Otterremo così un file chiamato file.gpg contenente il messaggio firmato e compresso.

- Per non comprimere il file, lasciandolo leggibile, si deve usare il comando:

```
gpg --clearsign [file]
```

- Per firmare un file binario o un archivio è meglio lasciare la firma in un file separato, per farlo usiamo il comando:

```
gpg -b [--armor] [file]
```

  Otterremo così un file chiamato file.sig contenente la firma del file.

- Per cifrare e allo stesso tempo firmare un file si può usare questo comando:

```
gpg [-u mittente] [-r destinatario] [--armor] --sign --encrypt [dati]
```

Utile per creare un "*armored ascii*"

Invio mail sicure

Uso di GPG – Verificare la firma di un file

- Per verificare la firma di un file:

```
gpg -verify [signed file]
```

Ad esempio:

```
micron@melindo micron $ gpg --verify file.gpg
gpg: Signature made Mon Jun  7 00:16:05 2004 CEST using DSA key ID
6D632BED
gpg: Good signature from "micron <micron@madlab.it>"
```

- Nel caso in cui si disponga di un file di tipo "*.sig*" oppure "*.asc*" si deve usare la seguente sintassi:

```
gpg --verify [sig/asc file] [file]
```

Ad esempio:

```
micron@melindo micron $ gpg --verify cmd.asc cmd
gpg: Signature made Mon Jun  7 00:18:46 2004 CEST using DSA key ID
6D632BED
gpg: Good signature from "micron <micron@madlab.it>"
```

Conservare alcuni dati

Soluzioni possibili

Possiamo agire in due modi:

- Usiamo un archvio cifrato con gpg
  - **Pro:**
    - ✔ Semplice realizzazione
    - ✔ Non richiede modifiche kernel
  - **Contro:**
    - ✔ Scomodo
    - ✔ Poco versatile
- Usiamo un container/device crittografato

Soluzione ottimale

Container/device crittografato

Cosa è un container crittografato?

Un container non è altro che un file (o un intero device) contenente un file system crittografato.

Pro:

- Comodo per conservare più file
- I vantaggi di avere un file system: permessi sui file, ...
- È visto come una partizione

Contro:

- Usa una cifratura a chiave simmetrica --> tutti gli svantaggi che ne derivano
- Richiede la presenza di alcuni componenti all'interno del kernel
- Si deve essere root del sistema (salvo configurazioni mirate)

Container/device crittografato

Soluzioni disponibili

Nell'ambiente GNU/Linux sono disponibili varie soluzioni:

- Bestcrypt
- Cryptoloop
- Loop AES
- Device mapper crypto

Container/device crittografato

Bestcrypt - caratteristiche

Soluzione **non open source** realizzata dalla Jetico, smette di funzionare dopo 30 giorni (?)

Pro:

- Usato in ambienti con alti requisiti di sicurezza
- Supporta cifratura devices
- Supporta molti algoritmi di cifratura
- Crea container leggibili anche da windows (se in fat/fat32)
- Innovativa feature: *hidden container*

Contro:

- Non è open source
- Non è compatibile con tutte le piattaforme (solo x86)
- Problemi con la serie 2.6 del kernel (fino alla release 1.5-5)
- Non supporta tutti i file system linux (supporta solo ext2, ext3, reiserfs, minix)

Container/device crittografato

Bestcrypt - avvio

Per prima cosa dobbiamo avviare bestcrypt:

```
gemini init.d # ./bcrypt start
```

- Sono caricati i seguenti moduli:

```
gemini init.d # lsmod
Module                  Size  Used by
bc_cast                18276  0
bc_rijn                33444  0
bc_idea                 8676  0
bc_3des                16484  0
bc_bf128               13604  0
bc_bf448               13732  0
bc_twofish             18752  0
bc_gost                 6884  0
bc_des                 15844  0
bc_blowfish            13604  0
bc                     16360  10
```

- Sono creati i seguenti devices:

```
/dev/bcrypt0
        ...
/dev/bcrypt15
```

Container/device crittografato

Bestcrypt – operazioni base

1) Creare un nuovo container:

```
bctool new -s [size] -a [algoritmo] [-d descrizione] [file/raw device]
```

2) Creare il file system del container:

```
bctool format -t [file system] [file]
```

3) Montare il container:

```
bctool mount -t [file system] [file] [mount point]
```

4) Smontare il container:

```
bctool umount [mount point]
```

Container/device crittografato

Bestcrypt – hidden container (1)

Applicando la *steganografia* a un container esistente si ottiene un *hidden container*.

Caratteristiche:

- È impossibile individuare la presenza di un hidden container.
- Usa una password (e volendo pure un file-system) diversa da quello del container ufficiale
- È ancora possibile accedere al container originale, dipende dalla password immessa al momento del mount.

Attenzione: ogni modifica fatta al primo container causerà la perdita definitiva di quello nascosto (utile per cancellare in maniera irreversibile dei dati in casi d'emergenza).

Container/device crittografato

Bestcrypt – hidden container (2)

Si consiglia di creare un *hidden container* all'interno di un container che abbia già alcuni dati (in modo da rendere ancora più difficile la sua individuazione).

Creare un hidden container:

```
bctool make_hidden [filename] [size]
```

Formattare l'hidden container* :

```
bctool format -t [file system] [file]
```

Montare l'*hidden container** :

```
bctool mount -t [file system] [file] [mount point]
```

* usare la password digitata al momento della creazione dell'hidden container

Container/device crittografato

Cryptoloop

Pro:

- Open source
- Supporto nativo all'interno del kernel (no patch con ramo 2.6)
- Supporta cifratura devices
- Supporta molti algoritmi di cifratura
- Multi piattaforma
- Supporta tutti i file system disponibili per Linux

Contro:

- Non supporta *hidden container*
- Non è compatibile con windows
- Vulnerabile ad attacchi ottimizzati con dizionario (scoperta recente)
- Il mount in loopback soffre di alcune limitazioni e bachi

Container/device crittografato

Crypto loop – Requisiti kernel

Per poter usare crypto loop di devono attivare le seguenti opzioni:

```
Device Drivers
|__
    Block devices
        |__
            Loopback device support (BLK_DEV_LOOP)
            |__
                Cryptoloop Support (BLK_DEV_CRYPTOLOOP)
```

Per quanto riguarda gli algoritmi scegliere tra quelli disponibili alla voce:

```
Cryptographic options
```

Compilare il tutto staticamente o modularmente.

NB: Per determinati scopi è richiesta la presenza statica e non modulare.

Container/device crittografato

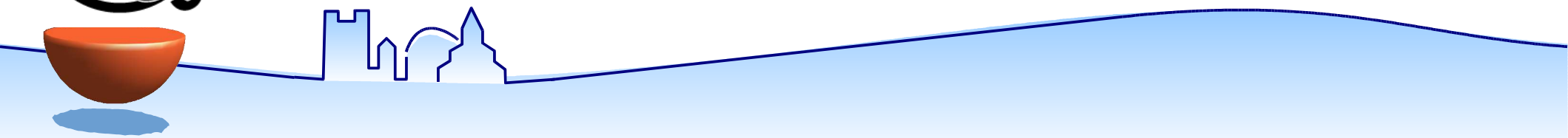
Crypto loop – Programmi in user space

Per poter usare crypto loop si devono compilare le ***util-linux***.

Devono essere aggiunte le patch per losetup e per il kernel della serie 2.6

```
$ wget http://www.kernel.org/pub/linux/utils/util-linux/util-linux-X.Y.tar.gz
$ tar xvzf util-linux-X.Y.tar.gz
$ cd util-linux-X.Y
$ patch -p1 < /dir_patchfile/losetup-combined.patch
$ patch -p1 < /dir_patchfile/util-linux-X.Y-kernel-2.6.patch
$ ./configure && make
# sudo install
```

Per varie distro è presente il pacchetto con le patch già applicate.



Container/device crittografato

Crypto loop – Creazione container (file)

Creiamo un file di 100 Mb che sarà il nostro container:

```
dd if=/dev/urandom of=~/container bs=1M count=100
```

- Il file è riempito di dati casuali.
- È ancora più difficile distinguere lo spazio libero da quello usato
- Maggiore sicurezza, meno vulnerabile ad attacchi di analisi

Associamo il container a un device in loopback:

```
losetup -e [algoritmo] /dev/loop[0-9] ~/container
```

- La password del container è creata in questo momento

Formattiamo il file system:

```
mkfs.[tipo fs] /dev/loop[0-9]
```

- Sono sconsigliati file system di tipo journaled
- La pagina di ram contenente il container può essere trasferita dalla ram alla swap, durante l'operazione non si tiene conto dell'ordine esatto di scrittura imposto dal journaling.

Container/device crittografato

Crypto loop – Creazione container (raw device)

Riempiamo di dati casuali il nostro device:

```
dd if=/dev/urandom of=/dev/[device]
```

- ✔ Maggiore sicurezza, meno vulnerabile ad attacchi “analitici”
- ✔ È ancora più difficile distinguere lo spazio libero da quello usato
- ✘ La procedura può richiedere un po' di tempo

Associamo il device cifrato a un device in loopback:

```
losetup -e [algoritmo] /dev/loop[0-9] /dev/[device]
```

- ➔ La password è creata in questo momento

Formattiamo il file system:

```
mkfs.[tipo fs] /dev/loop[0-9]
```

- ➔ Si possono usare tranquillamente file system di tipo journaled

Container/device crittografato

Crypto loop – mount e unmount del container

Montiamo il nostro container

```
mount -t [file system] /dev/loop[0-9] [mount point]
```

- Accertarsi d'avere eseguito *losetup*

Smontiamo il contianer:

```
umount [mount point]
losetup -d /dev/loop[0-9]
```

Nel caso in cui la password inserita con *losetup* sia sbagliata si ottiene il seguente errore:

```
mount: wrong fs type, bad option, bad superblock on /dev/loop0,
       or too many mounted file systems
```

- eseguire quindi:

```
losetup -d /dev/loop[0-9]
```

→ riprovare la procedura di mount

Container/device crittografato

Loop AES

Pro:

- Open source
- Supporta cifratura devices
- Multi piattaforma
- Supporta tutti i file system disponibili per Linux
- Supporta svariati algoritmi
- Supporta gpg (usi interessanti)
- Non vulnerabile ad attacchi ottimizzati con dizionario (supporta *key iterations* e *crypto salts* --> ***S2K***)

Contro:

- Installazione non immediata
- Vieta la presenza di alcuni moduli nel kernel
- Non supporta *hidden container*
- Non è compatibile con windows

Container/device crittografato

Loop AES – Requisiti kernel

Per poter usare *loop AES* si deve disattivare la seguenti opzione:

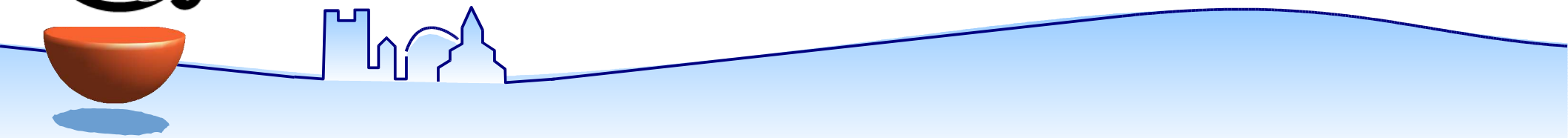
```
Device Drivers
|__
    Block devices
        |__
            Loopback device support (BLK_DEV_LOOP)
```

È invece raccomandata la presenza della seguente voce:

```
General setup
|__
    Enable loadable module support (MODULES)
        |__
            Automatic kernel module loading (KMOD)
```

Si può quindi compilare *loop AES*:

```
tar xvzf loop-AES-latest.tar.bz2
cd loop-AES-v2.1b && make && make install
```



Container/device crittografato

Loop AES – Programmi in user space

Anche loop AES richiede *util-linux*, ovviamente con apposite patch

```
wget http://ftp.cwi.nl/aeb/util-linux/util-linux-2.12a.tar.gz
tar xvzf util-linux-2.12.a.tar.gz
cd util-linux-2.12a
patch -p1 <../util-linux-2.12a.diff
CFLAGS=-O2 ./configure
make SUBDIRS="lib mount"
cd mount
install -m 4755 -o root mount umount /bin
install -m 755 losetup swapon /sbin
rm -f /sbin/swapoff && ( cd /sbin && ln -s swapon swapoff )
rm -f /usr/share/man/man8/ mount,umount,losetup,swapon,swapoff}.8.gz
install -m 644 mount.8 umount.8 losetup.8 /usr/share/man/man8
install -m 644 swapon.8 swapoff.8 /usr/share/man/man8
rm -f /usr/share/man/man5/fstab.5.gz
install -m 644 fstab.5 /usr/share/man/man5
mandb
```

NB: i requisiti in kernel ed user space rendono impossibile la coesistenza di *loop AES* e *crypto loop*

NB: non esistono pacchetti per le distro

Container/device crittografato

Loop AES – Funzionamento

Loop AES ha un funzionamento identico a *crypto loop*.

Le differenze principali sono date dalle sue nuove caratteristiche:

- *Supporto gpg:* la password del container può essere cifrata usando gpg

```
losetup -G [gpg home dir]
```

- *Key iteraction:* viene applicata la funzione di hashing *x* volte

```
losetup -C [iter counter]
```

- *Crypto salts:* prima dell'hasing alla password è aggiunto un valore casuale chiamato *pseed*

```
losetup -S [pseed]
```

Container/device crittografato

dm-crypt

Pro:

- Open source
- Supporta cifratura devices
- Multi piattaforma
- Supporta tutti i file system disponibili per Linux
- Supporta svariati algoritmi
- Supporto diretto nel kernel (a partire da 2.6.4)
- Non vulnerabile ad attacchi ottimizzati con dizionario (supporta *key iterations* e *crypto salts* --> ***S2K***)
- Non usa device in loopback (soffrono di alcuni bug)
- Compatibile con crypto loop
- Futura integrazione con LVM e EVMS (?!)

Contro:

- Supporta solo cifratura dei devices (no file in loopback)
- Non supporta *hidden container*
- Non è compatibile con windows

Container/device crittografato

dm-crypt – Requisiti kernel

Si devono attivare le seguenti opzioni:

```
Device Drivers
|__
    Multiple devices driver support (RAID and LVM) (MD)
        |__
            Device mapper support (BLK_DEV_DM)
                |_
                    Crypt target support (DM_CRYPT)
```

NB: non è richiesta la presenza di:

```
Device Drivers
|__
    Block devices
        |__
            Loopback device support (BLK_DEV_LOOP)
            |__
                Cryptoloop Support (BLK_DEV_CRYPTOLOOP)
```

La loro presenza non rappresenta un ostacolo.

Container/device crittografato

dm-crypt – Programmi in user space

dm-crypt richiede la presenza dei seguenti programmi:

- *device-mapper*, si possono trovare i pacchetti precompilati per varie distro.
- *cryptsetup*, lo si può scaricare da http://www.saout.de/misc/dm-crypt/

  Richiede:

  - libgcrypt 1.1.x
  - libdevmapper
  - hashlot (opzionale)

Container/device crittografato

dm-crypt – Creazione virtual device

Per cifrare un device

```
dd if=/dev/urandom of=/dev/[device]
```

- Il file è riempito di dati casuali.
- È ancora più difficile distinguere lo spazio libero da quello usato
- Maggiore sicurezza, meno vulnerabile ad attacchi analitici

Creiamo il device virtuale cifrato:

```
cryptsetup -y -c [algoritmo] create crypto /dev/[device]
```

NB: per avere una lista degli algoritmi a disposizione:

```
cat /proc/crypto
```

- È creato */dev/mapper/crypto*

Formattiamo il file system:

```
mkfs.[tipo fs] /dev/mapper/crypto
```

- Si possono usare tranquillamente file system di tipo journaled

Container/device crittografato

dm-crypt – Operazioni sul virtual device

Per avere informazioni sul device virtuale:

```
cryptsetup status /dev/mapper/[volume]
```

Smontare il device virtuale:

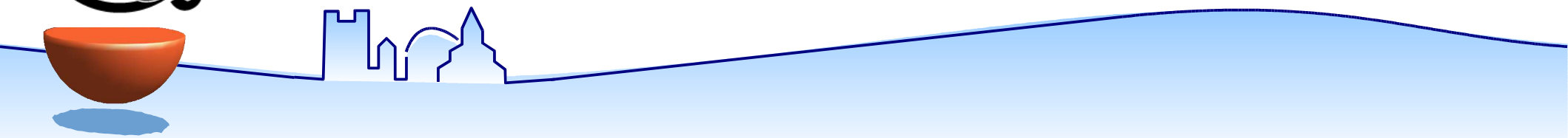
```
cryptsetup remove /dev/mapper/[volume]
```

Aggiornare il device virtuale:

```
cryptsetup reload /dev/mapper/[volume]
```

Ridimensionare il device virtuale:

```
cryptsetup resize /dev/mapper/[volume]
```



Intero sistema crittografato

Caratteristiche

Abbiamo la necessità di cifrare tutti i dati presenti sul sistema.

Pro:

- Totalmente inaccessibile, non si possono recuperare dati da partizioni non cifrate (*per esempio da* `/tmp`)
- Il sistema è completamente oscuro per un attaccante

Contro:

- Il sistema risulta più lento (a seconda delle sue caratteristiche hardware)
- Se la password viene smarrita l'intero sistema è "**perso**"

Intero sistema crittografato

Implementazione

Cifreremo **tutte le nostre partizioni** e utilizzeremo un sistema di autenticazione aggiuntivo via token.

Il sistema funziona nel seguente modo:

1) I devices sono cifrati usando chiavi di 54 caratteri casuali
2) Le chiavi sono conservate all'interno di una partizione cifrata di una chiave usb (*keychain*). La password del *keychain* è mnemonica
3) Il boot del sistema avviene da una partizione di boot non cifrata

Intero sistema crittografato

Pregi & Difetti

Pro:

1) Solo chi è in possesso del keychain e conosce la sua password può avviare il sistema
2) Grazie alle password di 54 caratteri casuali è impossibile praticare un attacco di tipo brute force sulle partizioni
3) Soluzione “ibrida”: anche i pc che non supportano il boot da usb possono aggirare il problema, godendo della medesima sicurezza.

Contro:

1) Senza il keychain il sistema non si può avviare

   --> conservarne alcune copie al sicuro
2) L'intera sicurezza del sistema dipende dalla password presente sul keychain --> (scegliere pwd adeguata, cambiarla regolarmente,...)

Intero sistema crittografato

Partizione di boot - requisiti

All'avvio il boot loader carica il kernel e avvia la procedura di init.
Il nostro sistema dovrà soddisfare questi requisiti:

- Il kernel si dovrà trovare in una partizione non cifrata
- Per montare la root dovremo disporre di alcuni programmi

Dovremo creare:

- Una partizione di boot non cifrata
- Un boot disk

Normale procedura, usare fdisk o simili. Formattare in ext2

Intero sistema crittografato

Boot disk – creazione (1)

Creiamo il file che conterrà il boot disk:

```
dd if=/dev/zero bs=1M count=5 of=/boot/initrd
```

Formattiamo il boot disk:

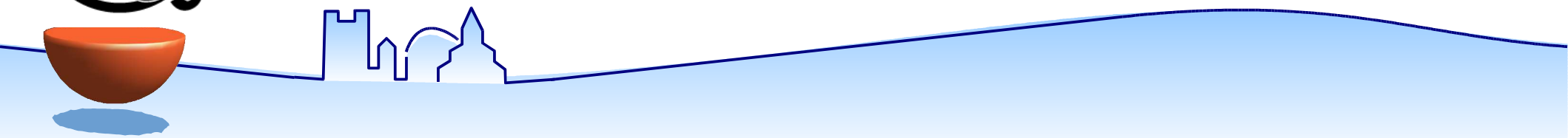
```
mke2fs -F -m0 -b 1024 boot/initrd
```

Montiamo il boot disk:

```
mount /boot/initrd -t ext2 -o loop=/dev/loop1 /mnt/initrd
```

Creaimo la struttura del sistema:

```
cd /mnt/initrd
mkdir -p {bin,dev,lib,mnt/{keychain,new-root},proc,usr/{sbin,lib},sbin}
```



Intero sistema crittografato

Boot disk - creazione (2)

Questi saranno programmi che ci serviranno:

- mount
- umount
- bash
- chroot
- killall
- pivot_root
- cryptsetup
- devfsd

Per trovare il percorso di un programma:

```
which program
```

Esempio:

```
melindo initrd # which cryptsetup
/usr/bin/cryptsetup
melindo initrd # cp /usr/bin/cryptsetup /mnt/initrd/usr/bin
```

Intero sistema crittografato

Boot disk - creazione (3)

Dovremo copiare anche le librerie a cui i vari programmi sono linkati.
Per trovare le librerie relative a un programma:

```
ldd `which program`
```

Esempio:

```
melindo initrd # ldd `which cryptsetup`
        libgcrypt.so.7 => /usr/lib/libgcrypt.so.7 (0x0ff93000)
        libnsl.so.1 => /lib/libnsl.so.1 (0x0ff5d000)
        libgpg-error.so.0 => /usr/lib/libgpg-error.so.0 (0x0ff39000)
        libdevmapper.so.1.00 => /lib/libdevmapper.so.1.00 (0x0ff13000)
        libpopt.so.0 => /usr/lib/libpopt.so.0 (0x0feea000)
        libc.so.6 => /lib/libc.so.6 (0x0fd9e000)
        /lib/ld.so.1 => /lib/ld.so.1 (0x30000000)
```

All'interno di */etc* ci saranno i seguenti files:

- *devfs.d* (è una dir)
- *devfsd.conf*

Intero sistema crittografato

Boot disk – init (1)

Dopo essersi caricato il kernel passa il controllo all'init.
Questo sarà il nostro script di init:

```
#!/bin/bash
counter=0

/bin/mount proc -t proc  /proc
/sbin/devfsd /dev 2> /dev/null

echo "waiting usb keychain..."
while ! [ -e /dev/sda2 ]; do
let 'counter += 1'
done

# Ask for a passphrase to open the keys (this prevents exposure of the keys in
# case the owner loses the keychain). Give the user three tries to get the
# passphrase right.

for ((FAILED=1, TRY=1; ($FAILED != 0) && (TRY <= 3); TRY++))
do
  if [ -e /dev/mapper/keychain ]; then
    /usr/bin/cryptsetup remove keychain
  fi
  /usr/bin/cryptsetup -c blowfish -s 448 --hash=ripemd160 create keychain /dev/sda2
  mount /dev/mapper/keychain /mnt/keychain
  FAILED=$?
done
```

Intero sistema crittografato

Boot disk – init (2)

```
if [ $FAILED -ne 0 ]; then
 echo "Sorry, you get only three attempts to guess the password."
 exit 1
fi

echo Keychain successfully mounted

echo Mounting partitions
#reading password from keychain
/usr/bin/cryptsetup -c aes -s 256 --hash=ripemd160 create root /dev/hda5 \
     < /mnt/keychain/melindo-root
/usr/bin/cryptsetup -c aes -s 256 --hash=ripemd160 create home /dev/hda6 \
     < /mnt/keychain/melindo-home

#remove the keychain
/bin/umount /mnt/keychain
/usr/bin/cryptsetup remove keychain
echo You can safely remove the keychain
```

Intero sistema crittografato

Boot disk – init (3)

```
/bin/mount /dev/mapper/root -t ext3 /mnt/new-root

# Pivot to the asset's root file system.
/bin/killall -9 devfsd
/bin/umount /proc

cd /mnt/new-root
/sbin/pivot_root . loader
/bin/mount devfs -t devfs /dev

# Pass control to init.
exec /usr/sbin/chroot . /sbin/init $* <dev/console >dev/console 2>&1
```

NB: ricordiamoci di modificare l'entry dei devices in */etc/fstab.*
Per esempio:

```
/dev/mapper/root    /     ext3       defaults            0 0
```

Intero sistema crittografato

Partizione di boot – ...in the end...

Smontiamo il boot disk:

```
umount /mnt/initrd
```

Controlliamo che in */boot* ci sia tutto l'occorrente:

- *vmlinux*
- *initrd*

Intero sistema crittografato

Configurazione kernel

All'interno del kernel dovremo attivare il suppporto al ram disk:

```
Device Drivers
|__
    Block devices
        |__
            RAM disk support (BLK_DEV_RAM)
                |_
                    Default RAM disk size (BLK_DEV_RAM_SIZE)
                |_
                    Initial RAM disk (initrd) support (BLK_DEV_INITRD)
```

Come dimensioni per il ram disk metteremo quelle del nostro boot disk

NB: nel kernel sono richieste anche le opzioni relative a dm-crypt, usb e usb-storage (ovviamente compilate statcamente)

Intero sistema crittografato

Bootloader - configurazione

Dovremo configurare anche il nostro boot loader:

- lilo:

```
image=/vmlinux
        initrd=/initrd
        root=/dev/ram0
        initrd-size=8192
        label=linux
        read-write
```

- grub:

```
title linux
root (hd0,0)
kernel /boot/vmlinuz root=/dev/ram0
initrd /boot/initrd
```

Intero sistema crittografato

Devices – creazione (1)

Creiamo la password per il nostro device:

```
head -c 45 /dev/urandom | uuencode -m - | head -n 2 | tail -n 1 > ~/[file]
```

Creiamo il device virtuale:

```
cryptsetup -v -c aes -s 256 --hash ripemd160 create [name] /dev/[device] < ~/[file]
```

Come solito prima di creare un device cifrato lo riempiremo di dati casuali.

Questa volta useremo openssl, al posto di /dev/urandom, come fonte di dati casuali.

```
openssl rand -out /dev/mapper/[name] [size]
```

- *size* è espresso in byte --> $1024^{\wedge 3}$= 1 Gb

ATTENZIONE: si consiglia di effettuare un backup dei propri dati!

Intero sistema crittografato

Devices – creazione (2)

Formattiamo il device:

```
mkfs.[tipo] /dev/mapper/[volume]
```

Montiamo il device:

```
mount /dev/mapper/[volume] -t [fs-type] [mount point]
```

Non resta altro che copiare i dati nel device, partendo dal backup.

Intero sistema crittografato

Keychain - creazione

Creiamo il device virtuale:

```
cryptsetup -v -c blowfish -s 448 --hash ripemd160 create keychain /dev/sda2
```

Prima di formattare un device cifrato lo riempiremo di dati casuali.

```
dd if=/dev/urandom of=/dev/mapper/keychain
```

Formattiamo il device:

```
mkfs.[tipo] /dev/mapper/keychain
```

Montiamo il device:

```
mount /dev/mapper/keychain -t [fs-type] /mnt/keychain
```

Copiamo le chiavi delle nostre partizioni sul keychain:

```
cp ~/[keys] /mnt/keychain/
```

Intero sistema crittografato

Considerazioni personali

- Usando il sistema interamente cifrato non ho notato notevoli rallentamenti.
- Volendo si può cifrare anche la swap (eccessivo al momento)
- La soluzione è comunque reversibile.
- Se particolarmente paranoici si può combiare regolarmente la password del keychain.
- dm-crypt è in continua evoluzione, ha un futuro roseo.
- È un modo come un altro per affidare la propria vita a una stupida chiave usb... ;)

Riferimenti

- **Gpg:**
  - http://www.gnupg.org/
- **Crypto-loop:**
  - http://www.kerneli.org
- **Loop AES:**
  - http://loop-aes.sourceforge.net/aespipe.README
- **dm crypto:**
  - http://www.saout.de/misc/dm-crypt/
- **Disk encryption howto:**
  - http://tldp.org/HOWTO/Disk-Encryption-HOWTO/
- **Encrypted Root Filesystem HOWTO:**
  - http://www.tldp.org/HOWTO/Encrypted-Root-Filesystem-HOWTO/
- **A Structured Approach to Hard Disk Encryption:**
  - http://www.sdc.org/~leila/usb-dongle/readme.html

La crittografia al servizio di Linux, conservare in maniera sicura i propri dati.

Domande ?