

NAME

Tomb – the Crypto Undertaker

SYNOPSIS

tomb [**options**] **command** [**arguments**]

DESCRIPTION

Tomb is an application to manage the creation and access of encrypted storage files: it can be operated from commandline and it can integrate with a user's graphical desktop.

Tomb generates encrypted storage files to be opened and closed using their associated keys, which are also protected with a password chosen by the user. To create, open and close tombs a user will need super user rights to execute the tomb commandline utility.

A tomb is like a locked folder that can be safely transported and hidden in a filesystem; it encourages users to keep their keys separate from tombs, for instance keeping a tomb file on your computer harddisk and its key file on a USB stick.

COMMANDS

- dig** Generates a file that can be used as a tomb and will occupy as much space as its desired initial size, the unlocked *.tomb* file can then be locked using a *key*. It takes a mandatory *-s* option which is the size in megabytes (MiB). Tombs are digged using low-quality random data (*/dev/urandom*).
- forge** Creates a new *key* and prompts the user for a *password* to protect its usage. This operation requires high quality random data (*/dev/random*) which can take quite some time to be gathered on a server: it works better on a desktop where the mouse can be moved around for entropy. The default cipher to protect the key is AES256, a custom one can be specified using the *-o* option, for a list of supported ciphers use *-v*. For additional protection against dictionary attacks on keys, the (experimental) *--kdf* option can be used when forging a key, making sure that the *tomb-kdb-pbkdf2* binaries in *extras/kdf* were compiled and installed on the system.
- lock** Initializes and locks an empty tomb (made with *dig*) using a key (made with *forge*), making it ready for usage. After this operation, the tomb can only be open in possession of the key and knowing its password. As in any other command requiring a key, the option *-k* should be used to specify a key file. The *-o* option can be used to specify the cipher specification: default is "aes-xts-plain64:sha256", old versions of Tomb used "aes-cbc-essiv:sha256". This operation requires root privileges to loopback mount, format the tomb (using LUKS and Ext4), then set the key in its first LUKS slot.
- open** Opens an existing *.tomb* (first argument) using a key (*-k*), if a second argument is given it will indicate the *mountpoint* where the tomb should be made accessible, else the tomb is mounted in a directory inside */media*. The option *-o* can be used to pass mount(8) options (default: *rw,noatime,nodev*).
- list** List all the tombs found open, including information about the time they were opened and the hooks that they mounted. If the first argument is present, then shows only the tomb named that way or returns an error if its not found. If the option *--get-mountpoint* is used then print a simple list of currently open tomb mountpoint paths.

- index** Creates or updates the search indexes of all tombs currently open: enables use of the *search* command using simple word patterns on file names. Indexes are created using *mlocate updatedb(8)* and stored in a file inside the tomb's root. To avoid indexing a specific tomb simply touch a *.noindex* file in its root.
- search** Searches through all tombs currently open for filenames matching one or more text patterns given as arguments. Search returns a list of files found in all open tombs on which the *index* command was run at least once.
- close** Closes a currently open tomb. If more tombs are open, the first argument should be used to specify the name of the tomb to be closed, or *all* to close all currently open tombs. This command fails if the tomb is in use by running processes (to force close, see *slam* below).
- slam** Closes a tomb like the command *close* does, but it doesn't fail even if the tomb is in use by other application processes: it looks for them and violently kills -9 each of them. This command may provoke unsaved data loss, but assists users to face surprise situations.
- passwd**
Changes the password protecting a key file specified using *-k*. The user will need to know the key's current password, then its content will be decoded and reencoded using the new one. This action can't be forced if the current password is not known. If the key file is broken (missing headers) this function also attempts its recovery.
- setkey** Changes the key file that locks a tomb, substituting the old one with a new one. Both the old and the new key files are needed for this operation and their passwords must be known. The new key must be specified using the *-k* option, the first argument should be the old key and the second and last argument the tomb file.
- resize** Increase the size of a tomb file to the amount specified by the *-s* option, which is the new size in megabytes (MiB). Full access to the tomb using a key (*-k*) and its password is required. Tombs can only grow and can never be made smaller. This command makes use of the *cryptsetup* *resize* feature and the *resize2fs* command: its much more practical than creating a new tomb and moving everything into it.
- engrave**
This command transforms a tomb key into an image that can be printed on paper and physically stored as backup, i.e. hidden in a book. It renders a QRCode of the tomb key, still protected by its password: a PNG image (extension *.qr.png*) will be created in the current directory and can be later printed (fits an A4 or Letter format). To recover an engraved key one can use any QRCode reader on a smartphone: save it into a file and then use that file as a key (*-k*).
- bury** Hides a tomb key (*-k*) inside a *jpeg image* (first argument) using *steganography*: the image will change in a way that cannot be noticed by human eye and hardly detected by data analysis. This option is useful to backup tomb keys in unsuspected places; it depends from the availability of *steghide*.
- exhume**
This command recovers from jpeg images the keys that were previously hidden into them using *bury*. Exhume requires a key filename (*-k*) and a *jpeg image* file (first argument) known to be containing a key. If the right key password is given, the key will be exhumed. If the password is not

known, it is very hard to verify if a key is buried in any image or not.

OPTIONS

-k <keyfile>

When opening a tomb, this option can specify the location of the key file to use. Keys are created with the same name of the tomb file adding a '.key' suffix, but can be later renamed and transported on other media. If <keyfile> is "-" (dash), it will read it from stdin.

-n Skip processing of post-hooks and bind-hooks if found inside the tomb. See the *HOOKS* section in this manual for more information.

-o Manually specify mount options to be used when opening a tomb instead of the default *rw,noatime,nodev*, i.e. to mount a tomb read-only (ro) to prevent any modification of its data. Can also be used to change the symmetric encryption algorithm for keys during *forge* operations (default *AES256*) or the LUKS encryption method during *lock* operations (default *aes-xts-plain64:sha256*).

-f Force flag, currently used to override swap checks, might be overriding more wimpy behaviours in future, but make sure you know what you are doing if you force an operation.

-s <MBytes>

When digging or resizing a tomb, this option must be used to specify the *size* of the new file to be created. Units are megabytes (MiB).

--kdf <iterations>

Activate the KDF feature against dictionary attacks when creating a key: forces a delay of <iterations> (integer multiplied by 10k) every time this key is used.

-h Display a help text and quit.

-v Display version and quit.

-q Run more quietly

-D Print more information while running, for debugging purposes

DEV MODE

--no-color

Suppress colors in console output (needed for string parsing by wrappers).

--unsafe

Enable using dev-mode arguments, i.e. to pass passwords from commandline options. This is mostly used needed for execution by wrappers and testing suite.

--use-urandom

Use an inferior quality random source to improve the speed of key generation at the cost of security (needed for the testing suite).

--sudo-pwd <string>

Use string as password when needed for privilege escalation via sudo.

--tomb-pwd <string>

Use string as password when needed on tomb.

--tomb-old-pwd <string>

Use string as old password when needed in tomb commands requiring multiple keys, like *passwd* or *setkey*.

-U Switch to this user ID when dropping privileges.

-G Switch to this group ID when dropping privileges.

-T Switch to this TTY terminal when dropping privileges.

HOOKS

Hooks are special files that can be placed inside the tomb and trigger actions when it is opened and closed; there are two kinds of such files: *bind-hooks* and *post-hooks* can be placed in the base root of the tomb.

bind-hooks

This hook file consists of a simple two column list of files or directories inside the tomb to be made directly accessible inside the current user's home directory. Tomb will use the "mount -o bind" command to bind locations inside the tomb to locations found in \$HOME so in the first column are indicated paths relative to the tomb and in the second column are indicated paths relative to \$HOME contents, for example:

```
mail          mail
.gnupg        .gnupg
.fmrc         .fetchmailrc
.mozilla      .mozilla
```

post-hooks

This hook file gets executed as user by tomb right after opening it; it should be a regular shell script, starting with a shell bang. Tomb executes this hook as user (dropping root privileges) and giving it two arguments: "\$1" is "open" or "close" depending from the tomb command given, "\$2" is the full path to the mountpoint where the tomb is open.

PRIVILEGE ESCALATION

The tomb commandline tool needs to acquire super user rights to execute most of its operations: to do so it uses sudo(8), while pinentry(1) is adopted to collect passwords from the user. Tomb executes as super user only when required.

To be made available on multi user systems, the superuser execution of the tomb script can be authorized for users without jeopardizing the whole system's security: just add such a line to */etc/sudoers*:

```
username ALL=NOPASSWD: /usr/local/bin/tomb
```

Password input is handled by the pinentry program: it can be text based or graphical and is usually configured with a symlink. When using Tomb in X11 it is better to use a graphical pinentry-gtk2 or pinentry-qt because it helps preventing keylogging by other X clients. When using it from a remote ssh connection it might be necessary to force use of pinentry-curses for instance by unsetting the DISPLAY environment var.

SWAP

On execution of certain commands Tomb will complain about swap memory on disk when that is present and *abort if your system has swap activated*. You can disable this behaviour using the *--force*. Before doing that, however, you may be interested in knowing the risks of doing so:

- During such operations a lack of available memory could cause the swap to write your secret key on the disk.
- Even while using an opened tomb, another application could occupy too much memory so that the swap needs to be used, this way it is possible that some contents of files contained into the tomb are physically written on your disk, not encrypted.

If you don't need swap, execute *swapoff -a*. If you really need it, you could make an encrypted swap partition. Tomb doesn't detect if your swap is encrypted, and will complain anyway.

EXAMPLES

- Create a 128MB large "secret" tomb and its keys, then open it:

```
tomb dig -s 128 secret.tomb

tomb forge secret.tomb.key

tomb lock secret.tomb -k secret.tomb.key

tomb open secret.tomb -k secret.tomb.key
```

- Open a Tomb using the key from a remote SSH shell, without saving any local copy of it:

```
ssh user@my.shell.net 'cat .secrets/tomb.key' | tomb open secret.tomb -k
```

- Open a Tomb on a remote server passing the unencrypted local key on stdin via SSH, without saving any remote copy of it:

```
gpg -d .secrets/tomb.key | ssh server tomb open secret.tomb -k cleartext
```

- Create a bind hook that places your GnuPG folder inside the tomb, but makes it reachable from the standard \$HOME/.gnupg location every time the tomb will be opened:

```
tomb open GPG.tomb -k GPG.tomb.key
echo ".gnupg .gnupg" > /media/GPG.tomb/bind-hooks
mv ~/.gnupg /media/GPG.tomb/.gnupg && mkdir ~/.gnupg
tomb close GPG && tomb open GPG.tomb -k GPG.tomb.key
```

- Script a tomb to launch the Firefox browser every time is opened, keeping all its profile data inside it:

```
tomb open FOX.tomb -k FOX.tomb.key
cat <<EOF > /media/FOX.tomb/post-hooks
#!/bin/sh
if [ "$1" = "open" ]; then
    firefox -no-remote -profile "$2"/firefox-pro &
fi
EOF
chmod +x /media/FOX.tomb/post-hooks
```

- Script a tomb to archive Pictures using Shotwell, launching it on open:

```
tomb open Pictures.tomb -k Pictures.tomb.key
cat <<EOF > /media/Pictures.tomb/bind-hooks
Pictures Pictures
EOF
cat <<EOF > /media/Pictures.tomb/post-hooks
#!/bin/sh
if [ "$1" = "open" ]; then
    which shotwell > /dev/null
    if [ "$?" = "0" ]; then
        shotwell -d "$2"/Pictures/.shotwell &
```

```
fi
fi
EOF
chmod +x /media/Pictures.tomb/post-hooks
```

BUGS

Please report bugs on the Github issue tracker at <https://github.com/dyne/Tomb/issues>

One can also try to get in touch with developers via the #dyne chat channel on <https://irc.dyne.org>.

AUTHORS

Tomb is designed, written and maintained by Denis Roio aka Jaromil.

Tomb includes code by Anathema, Boyska, Hellekin O. Wolf and GDrooid.

Tomb's artwork is contributed by Jordi aka Mon Mort and Logan VanCuren.

Gettext internationalization and Spanish translation is contributed by GDrooid, French translation by Hellekin, Russian translation by fsLeg, German translation by x3nu.

Testing, reviews and documentation are contributed by Dreamer, Shining the Translucent, Mancausoft, Asbesto Molesto, Nignux, Vlax, The Grugq, Reiven, GDrooid, Alphazo, Brian May, TheJH, fsLeg, Joel-Mon and the Linux Action Show!

Cryptsetup was developed by Christophe Saout and Clemens Fruhwirth.

COPYING

This manual is Copyright (c) 2011-2014 by Denis Roio <jaromil@dyne.org>

This manual includes contributions by Boyska.

Permission is granted to copy, distribute and/or modify this manual under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation. Permission is granted to make and distribute verbatim copies of this manual page provided the above copyright notice and this permission notice are preserved on all copies.

AVAILABILITY

The most recent version of Tomb sourcecode and up to date documentation is available for download from its website on <http://tomb.dyne.org>.

SEE ALSO

cryptsetup(8)

GnuPG website on <http://www.gnupg.org>

DM-Crypt website on <http://www.saout.de/misc/dm-crypt>

LUKS website, <http://code.google.com/p/cryptsetup>