Некоторые особенности GPG подписей, часть 1.

GNU Privacy Guard (GnuPG, GPG) - это инструмент асимметричного шифрования.

Для чего он может понадобиться? Для шифровки и дешифровки, подписей и проверки подписей. По сути инструмент заменяет подпись ручкой на бумаге, только электронный.

Вот несколько примеров, некоторые из которых далее будут рассмотрены более подробно:

- git — верификация комитов, что это именно ваш коммит и он не скомпрометирован.
- Подпись пакетов локального или удалённого репозитория.
- Шифровка/дешифровка сообщений.

Примечание: Если что-то забыли или не знаете — не стесняйтесь help и man. Адекватно и понятно, даже на английском языке. Не забывайте, что внутри утилиты также есть help.

Начнём с того, что уже в любом Linux присутствуют 2 инструмента, таких как gpg и gpg2. Первый из них устаревший. Однако, оба работают со всеми командами абсолютно одинаково. Для git в большинстве статей рекомендуют настраивать на работу с gpg2.

Первое что может быть крайне удобным — это ключ для работы с утилитой gpg: «--homedir». Таким образом нам не обязательно вообще держать ключ в самой ОС.

Создадим ключ. Лучше всего использовать полную генерацию для подробной первоначальной настройки. Дело в том, что большинство git репозиториев рекомендую длину всех ключей 2048.

$ gpg --full-generate-key --expert

После чего его необходимо настроить (в режиме эксперта по желанию):

$ gpg --edit-key UserName --expert

$ uid 1 # Выбор идентификатора.

По окончании работы с идентификатором обязательно повторно набрать команду, чтобы сбросить селектор. Также и с командой key.

$ trust # уровень доверия.

Рекомендую сразу ставить 3. Именно данный уровень, а не 5 — абсолютное доверие. В дальнейшем он понадобиться при создании собственного пакета с публичными ключами. Дело в том, что пользовательский уровень доступа на публичных, а также удалённых серверах — это 3. Если он нужен только локально, т.е. для себя (например верификация пакетов git), то выбирайте любой уровень доверия.

Пока вы находитесь внутри утилиты gpg не забудьте настроить уровни доверия для всех подключей, если таковые имеются.

$ list

$ key 1

$ trust

...

$ key 1

$ key 2

...

И я не зря упомянул подключи. Дело в том, что вы можете использовать один и тот же идентификатор пользователя и пароль, но разные подключи для разных операций. Например, один подключ только для подписей, а другой только для шифрования. При создании подключа обязательно указываем его тип и не забываем также настроить для него уровень доверия.

Дело в том, что вы можете использовать разные подключи для подписи пакетов в разных репозиториях, что сильно повысит их безопасность.

Чтобы использовать нужный подключ, просто наберите «-u».

Например: $ gpg --homedir /home/mikl/002/ -u 94DB5679387284B0 -b 001.txt

Или: $ find homemikl/custom_repo/x86_64/ -type f -exec gpg2 --homedir /home/mikl/002/ -u 94DB5679387284B0 -b {} \;

Также стоит упомянуть и о сроке действия ключа и подключей. Пока вы не отозвали ключ, срок его годности можно бесконечно продлевать. Не забывайте только обновлять его на публичном сервере ключей. Как только ключ был отозван — изменить ничего уже будет нельзя. К сожалению утилита не имеет команд для того чтобы вернуть ключу доступ. После этого вам придётся создавать ключи заново (gpg --gen-key). Только если

вы не делали резервных копий (смотрите далее) и не отправляли ключ на публичный сервер.

По окончании работы внутри утилиты, набираем: $ save

$ gpg -k # Посмотреть публичные ключи

$ gpg -K # Посмотреть секретные ключи

$ gpg --list-secret-keys --keyid-format LONG # Перечислить ключи для которых есть как открытый так и закрытый ключ.

С помощью последней команды можно найти короткую версию идентификатора любого ключа во 2 столбце после алгоритма и слеша. Самый первый короткий идентификатор и есть первичный ключ, который можно использовать где-то вместо других вариантов выбора ключа. Например при отправке ключа на публичный сервер.

Если вы готовы зарегистрируйте публичный ключ в публичном сервере ключей. Прямо каламбур.

$ gpg --keyserver pgp.mit.edu --send-keys <идентификатор> # можно короткую версию.

Теперь создадим резервные копии ключей. Не мне объяснять вам зачем нужны резервные копии. Они также повысят ваши шансы при различных сбоях и косяках.

$ gpg --armor --output mykey.gpg --export UserName

Публичные ключи. Сразу после output пишется полная директория и названия файла куда он будет сохранён. После export пишется либо Имя пользователя ключа, либо e-mail, либо идентификатор ключа.

Аналогичная, но более понятная команда:

$ gpg -a --export UserName > public_key.gpg

$ gpg -a --export-secret-keys UserName > secret_key # Экспорт секретного ключа

$ gpg -a --export-secret-subkeys UserName > secret_subs.gpg # Экспорт секретных подключей

$ gpg --export-ownertrust > trust.conf # Экспорт уровня доверия ключей. Нужен для создания собственного пакета ключей или при обмене с другими пользователями, а также для собственного репозитория.

$ gpg -a --gen-revoke UserName > revocation_cert.gpg

Генерировать сертификат отзыва ключа. Если что-то пойдёт не так он пригодится, чтобы прекратить действие ваших ключей.

Как только вы попытаетесь его импортировать (добавить) с введением пароля — все ваши GPG ключи для указанного идентификатора будут отозваны. Пока только локально. Если вы не отправляли ключи на публичный сервер - их можно удалить с пк, вместе с публичными и импортировать заново. Потому что вы создавали резервную копию в момент когда ключи были действующими.

После этого можно повторно экспортировать секретные и публичные ключи. В таком случае будет сохранён новый срок их годности. После чего не забудьте обновить на публичном сервере ключей.

Ключи обновляются либо с помощью сервера ключей, либо постепенно, т.е. пофайлово. Например: если у вас есть старые

ключи, которые действуют 3 года, а прошло только 2. И вдруг владелец отозвал их. Случайно обновились или добавили новые — удалите их и импортируйте старые ключи заново. Без обновлений или с их резервными копиями вы можете продолжать с ними работать, словно ключи никогда не были отозваны.

Импорт ключей, секретных ключей, секретных подключей. Аналогично происходит импорт сертификата отзыва ключей для прекращения действия последних.

$ gpg --import elseworld.gpg

$ gpg --import-ownertrust trust.conf # Импорт уровней доверия ключей

Создать отделенную, а также проверить подпись:

$ gpg -b 001.txt

$ gpg --verify 001.txt.sig 001.txt

О настройках git и Github, а также других интересных особенностях смотрите в следующей статье.

Некоторые особенности GPG подписей, часть 2.

Продолжение предыдущей статьи.

Для того, чтобы настроить ваш git на созданную подпись, выполните следующие команды.

$ git config --global user.signingkey <Идентификатор> # использовать для подписи указанную

$ git config --global gpg.program gpg2 # Использовать в git - gpg2

$ git config --global commit.gpgsign true

Включить глобальное подписывание комитов. Без данной команды подписывать комиты придётся вручную. Например: $ git commit -S -m "Update"

Перед тем, как что-либо подписывать убедитесь, что e-mail и имя пользователя в настройках самого git, github, а также в самих GPG ключах указаны абсолютно одинаковые. Иначе при верификации комитов получите «Unverifed».

Не забудьте импортировать публичный ключ в настройках самого github.

Для того, чтобы иметь возможность проверять подписи комитов у пользователя должны быть импортированы публичные ключи доступа (gpg —import, GPG). Не путайте с ключами pacman (pacman-key, PGP).

$ git log --show-signature -1

$ git merge --verify-signatures non-verify

Если вам необходимо устанавливать пакеты из репозитория без проверок ключи должны быть добавлены в pacman и обязательно локально подписаны.

Посмотрим последнее на примере ctlos.

sudo pacman-key --init

sudo cp ctlos{.gpg,-revoked,-trusted} /usr/share/pacman/keyrings/

sudo pacman-key --add /usr/share/pacman/keyrings/ctlos.gpg

sudo pacman-key --lsign-key 50417293016B25BED7249D8398F76D97B786E6A3

sudo pacman-key --populate archlinux ctlos # Перезагрузить ключи из указанных связок

sudo pacman-key —refresh-keys # обновление указанных или всех ключей с сервера ключей

Как только у вас есть есть все необходимые данные можно создавать отдельный репозиторий со статичными данными для публичных ключей, чтобы конечный пользователь мог их найти и воспользоваться ими для верификации данных.

Для настройки Github в качестве ftp сервера для ваших пакетов – зайдите в настройку каждого вашего подобного репозитория и в самом низу на на главной вкладке «Options» найдите «GitHub Pages». Выберите ветку (например master), дерктрорию (лучше / root) и нажмите «Save». После этого у вас появится ссылка на ваш репозиторий в виде https ссылки при установленной ниже галочке «Enforce HTTPS» или в виде http ссылки без последней.

Чтобы использовать удалённый репозиторий с верификацией пакетов - аналогично стандартным реп-ям ArchLinux — надо сделать ещё несколько шагов. Ключи можно добавлять в систему несколькими способами: скриптом, запрашивать с сервера ключей, устанавливать из пакета с публичными ключами, скачивать с аналогичного репозитория...

В любом случае придётся генерировать конечные пакеты с ключами и заливать их в каждый репозиторий отдельно - для того, чтобы конечный пользователь своевременно обновлялся.

Поэтому создаём несколько файлов, некоторые из которых предназначаны только для ускорения доступа. Не забудьте придумать название для ваших публичных ключей. Любое, главное короткое. У ctlos — соответственно — ctlos.gpg, ctlos-revoked, ctlos-trusted, ctlos--keyring.install.

- elseworld.gpg — Публичные ключи.
- elseworld-id.txt — Полный глобальный первичный идентификатор ключа. Необязателен. Файл упрощает жизнь при использовании доп. скриптов, чтобы каждый раз при изменении ключа лишний раз в них не лазить.
- elseworld-keyring.install — инсталятор ключей, используется в PKGBUILD, без него он не заработает.
- Elseworld-revoked — Идентификатор отзыва ключа. Используется только в том случае, когда ваш GPG ключ отзывается, прекращает своё действие. Разумеется в публичном ключе должен быть новый ключ, который действует. Так делают в стандартном пакете «archlinux-keyring». Чаще всего этот файл не используется, но должен присутствовать хотя бы пустым.
- Elseworld-trusted — таблица доверия ключей. Помните мы экспортировали уровни доверия ключей? Вот сюда этот файлик и направляется. Уберите только строки с решетками.
- install-keyring.sh — здесь скрипт для скачивания и импорта публичных ключей в pacman-key. Чтобы не вводить все команды вручную, если вдруг придётся. Скачиваем скрипт и запускаем от имени суперпользователя. Можно даже вместо установочного пакета. Значения не имеет. Шикарно.
- Makefile — сборочник ключей, без которых не может быть их правильной сборки в PGKBUILD.
- PKGBUILD

- README.md
- md5sums.txt — создавал 2 раза. 1-й когда редактировал PKGBUILD, 2-й когда полностью завершил работу со всеми файлами. На всякий случай. Не обязательный файл. Но так, верифицированный репозиторий с публичными ключами будет ну совсем крутой. Только удалите из него 2 строки: index.html и сам md5sums.txt.

Содержимое Makefile:

```
V=20200805

PREFIX = /usr

install:

install -dm755 $(DESTDIR)$(PREFIX)/share/pacman/keyrings/

install -m0644 elseworld{.gpg,-trusted,-revoked} $(DESTDIR)$
(PREFIX)/share/pacman/keyrings/

uninstall:

rm -f $(DESTDIR)$(PREFIX)/share/pacman/keyrings/elseworld{.gpg,-
trusted,-revoked}

rmdir -p --ignore-fail-on-non-empty
$(DESTDIR)$(PREFIX)/share/pacman/keyrings/
```

Содержимое elseworld-keyring.install:

```
post_upgrade() {

if usr/bin/pacman-key -l >/dev/null 2>&1; then
```

usr/bin/pacman-key --populate elseworld

fi

}

post_install() {

if [-x usr/bin/pacman-key]; then

post_upgrade

fi

}

Содержимое install-keyring.sh:

#!/bin/bash

maximalisimus keyring

#

sudo pacman-key --init

wait

wget https://maximalisimus.github.io/elseworld-keyring/elseworld{.gpg,-revoked,-trusted,-id.txt}

wait

sudo mv elseworld{.gpg,-revoked,-trusted} /usr/share/pacman/keyrings/

```
wait

sudo pacman-key --add /usr/share/pacman/keyrings/elseworld.gpg

wait

ewid=$(cat elseworld-id.txt)

wait

sudo pacman-key --lsign-key ${ewid}

wait

sudo pacman-key --populate archlinux elseworld

wait

sudo pacman-key --refresh-keys

wait

rm -rf elseworld-id.txt

exit 0
```

Содержимое PKGBUILD (А вот и те самые md5sums пригодились для верификации сборки):

```
# Maximalisimus keyring gpg-key

pkgname=elseworld-keyring

pkgver=stable

pkgrel=1
```

```
pkgdesc='elseworld PGP keyring'

arch=('any')

url='https://github.com/maximalisimus/elseworld-keyring'

license=('GPL')

install="${pkgname}.install"

source=('Makefile'

'elseworld.gpg'

'elseworld-revoked'

'elseworld-trusted')

md5sums=('547c89ef8b1c509a438dc580b08762d5' # Makefile

'284617f80ad66083b12b5cd9fa9d0ae4' # *.gpg

'd41d8cd98f00b204e9800998ecf8427e' # *-revoked

'077744c6332a8619d0c2e8d1f68956e3') # *-trusted

package() {

cd "${srcdir}"

make PREFIX=/usr DESTDIR=${pkgdir} install

}
```

Чтобы на Github Pages появились статические данные нужен ещё пакет «apindex»:

$ apindex .

Обратите внимание на точку — для текущей и вложенных директорий. Помощи у утилиты нет.

Чтобы не отображать скрытую папку «.git» для текущего репозитория выполните команду:

$ find ./.git/ -type f -iname "index.html" -exec rm -rf {} \;

Также вручную отредактируйте один из самых верхний файлов «index.html» и уберите из него часть html таблицы с данной папкой («.git»).

Всё, собираем (makepkg -s), лишние папки (src и pkg) удаляем и заливаем на github. Созданый пакет ключей также заливаем во все свои репозитории.

Вот теперь можно подписать все пакеты своих удалённых репозиториев отделённой подписью ($ find ./ -type f -exec gpg2 -b {} \;) и заливать всё на сервер.

Маленький бонус. Представляю вам свой скрипт для обновления баз данных пакетов в одном из удалённых репозиториев.

```
#!/bin/bash

#

rm -rf *.sig

wait

find ./ -type f -exec gpg2 -b {} \;
```

```
wait

rm -rf ./index.html.sig

wait

repo-add -n -R aur-package-manager.db.tar.gz *.pkg.tar{.xz,.zst}

wait

rm -rf aur-package-manager.db

wait

cp -f aur-package-manager.db.tar.gz aur-package-manager.db

wait

##optional-remove for old repo.db##

# rm -rf *gz.old{,.sig}

wait

apindex .

wait

exit 0
```

Обратите внимание на вот эту конструкцию: {.xz,.zst}.

Еси у вас в реп-ии только пакеты xz — то скрипт завершится с ошибкой. Тоже самое при zst. В таком случае указывайте конкретное окончании - *.pkg.tar.xz, *.pkg.tar.zst.

После этого можно всё заливать и пользоваться:

git push -u origin master

Всем добра и удачи!