GnuPG (GPG) Cheatsheet

A comprehensive guide to using GnuPG (GPG) for encryption, signing, and key management. This cheatsheet covers essential commands and workflows for securing your communications and data.

Key Management

Generating Keys

Generate a new key pair:

```
gpg --gen-key
```

This command starts an interactive process to generate a new key pair. You'll be prompted for various options like key type, key size, and expiration date.

Generate a new key pair with dialogs for all options:

```
gpg --full-gen-key
```

Provides more detailed options during key generation, such as selecting the key algorithm and curve.

Batch Key Generation (without interaction):

```
gpg --batch --gen-key <(echo '%no-protection\n%transient-key\nKey-Type: Ed25519\nName-Real: Your Name\nName-Email: your.email@example.com\nExpire-Date: 0\n%commit\n')
```

Automates key generation, useful for scripting. Replace `Your Name` and `your.email@example.com` with your actual information.

Listing Keys:

```
gpg --list-keys        # List public keys
gpg --list-secret-keys # List secret keys
gpg -k                 # Short form for list public keys
gpg -K                 # Short form for list secret keys
```

These commands display the keys in your keyring. Public keys are used to encrypt messages to you, while secret keys are used to decrypt messages and sign documents.

Listing Keys with Fingerprints:

```
gpg --fingerprint <KEY_ID>
```

Display the fingerprint of a specific key. Very important for verifying key identity with others.

Exporting and Importing Keys

Exporting Keys:

```
gpg -o key.gpg --export <KEY_ID>
```

Exports the key in binary format.

Exporting Keys in ASCII:

```
gpg -o key.asc --armor --export <KEY_ID>
```

Exports the key in an ASCII armored format, suitable for sharing via text.

Importing Keys:

```
gpg --import key.gpg
gpg --import key.asc
```

Imports keys from a file.

Importing with Merge-Only Option:

```
gpg --import key.asc --import-options merge-only
```

Only updates existing keys in your keyring, ignoring new keys.

Exporting Secret Key:

```
gpg -o secret-key.gpg --export-secret-key <KEY_ID>
```

Exports the secret key (keep this secure!). Add `--armor` for ASCII format.

Considerations for Secret Key Export:

- **Security:** Treat the exported secret key with extreme care.
- **Backup:** Export for backup purposes, storing it securely offline.
- **Transfer:** Use secure methods (e.g., encrypted storage) if transferring the secret key.

Key Servers

Importing Keys from a Keyserver:

```
gpg --receive-keys <KEY_IDS>
```

Downloads keys from a keyserver.

Uploading Keys to a Keyserver:

```
gpg --send-keys <KEY_IDS>
```

Uploads your public key to a keyserver.

Refreshing Keys from a Keyserver:

```
gpg --refresh-keys
```

Updates keys in your keyring from a keyserver.

Searching for Keys on a Keyserver:

```
gpg --search-keys "<SEARCH STRING>"
```

Searches for keys on a keyserver.

Specifying a Keyserver:

```
gpg --keyserver <URL> ...
```

Overrides the default keyserver. Add to `~/.gnupg/gpg.conf` for persistent configuration.

Encryption and Decryption

Public Key Encryption

Encrypting a File:

```
gpg -e -o secret.txt.gpg -r <RECIPIENT>
secret.txt
```

Encrypts `secret.txt` for the specified recipient, creating `secret.txt.gpg`.

Specifying Recipient Options:

```
gpg -e -r <KEY_ID> ...
gpg -e -r "Bez" ...
gpg -e -r "bezalelhermoso@gmail.com" ...
```

Use key ID, name, or email to specify the recipient.

Encrypting for Multiple Recipients:

```
gpg -e -r <RECIPIENT> -r <ANOTHER_RECIPIENT>
... secret.txt
```

Encrypts the file so that multiple recipients can decrypt it.

Important Notes:

- Omitting `-o|--output` creates `<ORIGINAL_FILENAME>.gpg`.
- Public key encryption requires the recipient's public key.

Symmetric Encryption

Encrypting with a Shared Key:

```
gpg --symmetric secret.txt
# or
gpg -c secret.txt
```

Encrypts the file using a passphrase, prompting for it during encryption. Anyone with the passphrase can decrypt the file.

Decryption

Decrypting a File:

```
gpg -d -o secret.txt secret.txt.gpg
```

Decrypts `secret.txt.gpg` into `secret.txt`.

Decrypting to Standard Output:

```
gpg -d secret.txt.gpg
```

Prints the decrypted content to standard output (terminal).

Passphrase Prompt:

For symmetric encryption, you'll be prompted for the passphrase.

Important Notes:

- Omitting `-o|--output` prints the output to stdout.

Signing and Verification

Signing Files

Creating a Detached Signature:

```
gpg -o file.txt.sig -b file.txt
```

Creates a detached signature file (`file.txt.sig`) for `file.txt`.

Creating an Integrated Signature:

```
gpg -o signed-file.txt.gpg -s file.txt
```

Creates an integrated signature, resulting in a binary file (`signed-file.txt.gpg`).

Signing and Encrypting:

```
gpg -s -o secret.txt.gpg -r <RECIPIENT> secret.txt
```

Signs the file while encrypting it.

Clearsigning a File:

```
gpg --clearsign file.txt
```

Creates a human-readable signature embedded within the file (creates `file.txt.asc`).

Verifying Signatures

Verifying a Detached Signature:

```
gpg --verify file.txt.sig file.txt
```

Verifies the signature file (`file.txt.sig`) against the original file (`file.txt`).

Verifying an Integrated Signature:

```
gpg --verify signed-file.txt.gpg
```

Verifies an integrated signature.

Verifying a Clearsigned File:

```
gpg --verify file.txt.asc
```

Verifies a clearsigned file.

Viewing Content of Signed File:

```
gpg -d signed-file.txt.gpg
```

Decrypts and displays the content of a signed file.

Advanced Usage and Troubleshooting

Trusting Keys

Trusting a Key Interactively:

```
gpg --edit-key <KEY_ID>
```

In the interactive prompt:

```
gpg> trust
gpg> save
```

Sets the level of trust you have in a key. This helps GPG decide if signatures from this key are valid.

Using Email/Name instead of Key ID:

You can often use the owner's email or name (or part thereof) instead of the key ID for `--edit-key`.

Trust levels:

- 1: I don't know or won't say
- 2: I do NOT trust
- 3: I trust marginally
- 4: I trust fully
- 5: I trust ultimately

Managing GPG Components

Listing Components:

```
gpgconf --list-components
```

Lists all GPG components.

Killing a Component:

```
gpgconf --kill <COMPONENT>
```

Kills a specific component (e.g., `gpgconf --kill dirmngr`).

Killing All Components:

```
gpgconf --kill all
```

Kills all running GPG components.

Restarting GPG Agent:

```
gpgconf --launch gpg-agent
```

Restarts the GPG agent, which manages secret keys.

Parsing Keyring Data

Using Colon-Separated Output:

```
gpg -k --with-colons
```

Produces output that is easily parsed with tools like `awk` and `grep`.

Quick Reference for Fields:

Refer to the GnuPG documentation for detailed explanations of each field. Common fields include Record Type, Validity, Key Length, Key ID, Creation Date, and User ID.

Troubleshooting

"No secret key" error:

Ensure the correct secret key is present in your keyring and that the GPG agent is running.

Signature verification failed:

Verify that you have the correct public key for the signer and that the original file hasn't been altered.

GPG agent issues:

Try restarting the GPG agent using `gpgconf --kill gpg-agent` followed by `gpgconf --launch gpg-agent`.

Keyserver errors:

Try a different keyserver or check your network connection.