GPG/GnuPG

Complete Encryption & Signing Guide

Modern Security Reference

Master GNU Privacy Guard for secure communications

Shell-Agnostic Commands for Modern Workflows

Version: 2.0
Updated: November 5, 2025
Focus: Encryption • Digital Signatures • Key Management

End-to-end encryption reference
for secure communications and authentication

Contents

1 What is GPG?

Information

GNU Privacy Guard (GPG) is a free implementation of the OpenPGP standard that enables you to encrypt and sign data and communications.

Core Features:
- End-to-end encryption for files and messages
- Digital signatures to verify authenticity
- Public/private key pair management
- Symmetric encryption using passwords
- Integration with email clients, Git, and other tools

2 Installation & Setup

2.1 Install GPG on macOS

Install the native ARM64 version optimized for Apple Silicon:

```
# Install via Homebrew
brew install gnupg

# Verify installation
gpg --version

# Install additional tools
brew install pinentry-mac
```

2.2 Initial Configuration

Configure GPG for optimal macOS integration:

```
# Create GPG directory if it doesn't exist
mkdir -p ~/.gnupg
chmod 700 ~/.gnupg

# Configure pinentry for macOS
echo "pinentry-program /opt/homebrew/bin/pinentry-mac" >> ~/.gnupg/gpg-
    agent.conf

# Restart GPG agent
gpgconf --kill gpg-agent
gpgconf --launch gpg-agent
```

Pro Tip

Shell Configuration: Add `export GPG_TTY=$(tty)` to your shell's RC file (`.bashrc`, `.zshrc`, etc.) to prevent "inappropriate ioctl" errors.

3 Security Fundamentals

Warning

Critical security practices when using GPG:

1. **Never share private keys** – Only distribute public keys
2. **Use strong passphrases** – Minimum 20 characters with complexity
3. **Regular key rotation** – Generate new keys yearly for high-security needs
4. **Verify fingerprints** – Always verify through separate channels
5. **Backup securely** – Store encrypted backups in multiple locations
6. **Set expiration dates** – Force regular security review
7. **Generate revocation certificates** – Prepare for key compromise scenarios

4 Key Management

4.1 Generate Keys

Creating your public/private key pair is the foundation of GPG encryption:

```
# Generate a new key pair (interactive)
gpg --full-generate-key

# Quick generate with defaults (RSA 3072-bit)
gpg --quick-generate-key "Your Name <email@example.com>"

# Generate with specific algorithm
gpg --quick-generate-key "Your Name <email@example.com>" rsa4096 sign,encr
```

Note

Key Algorithm Recommendations:

- **RSA 2048** – Minimum for basic security (faster)
- **RSA 3072** – Recommended default (good balance)
- **RSA 4096** – Maximum security (slower operations)
- **ECC (Curve25519)** – Modern alternative, shorter keys

4.2 List & Export Keys

View and share your keys:

```
# List public keys
gpg --list-keys
gpg -k  # Short version

# List secret keys
gpg --list-secret-keys
gpg -K  # Short version

# Export public key (ASCII format)
gpg --armor --export your@email.com > public_key.asc

# Export secret key (be careful!)
gpg --armor --export-secret-keys your@email.com > private_key.asc

# Export to keyserver
gpg --keyserver hkps://keys.openpgp.org --send-keys YOUR_KEY_ID
```

Warning

Private Key Export Warning: Exporting private keys is extremely dangerous. Only export for secure backup purposes. Store exported key in encrypted storage and never send via email or unencrypted channels.

4.3 Import Keys

Add keys to your keyring:

```
# Import public key from file
gpg --import public_key.asc

# Import from keyserver
gpg --keyserver hkps://keys.openpgp.org --receive-keys KEY_ID

# Import and trust automatically
echo "FINGERPRINT:6:" | gpg --import-ownertrust
```

Information

Trust Levels:

- **Unknown (0)** – Not yet evaluated
- **Never (1)** – Key is not trusted
- **Marginal (2)** – Some trust
- **Full (3)** – Fully trusted
- **Ultimate (4)** – Your own keys

4.4 Delete Keys

Remove keys from your keyring:

```
# Delete public key
gpg --delete-key "user@example.com"

# Delete private key (DANGEROUS!)
gpg --delete-secret-key "your@email.com"

# Delete both (requires two confirmations)
gpg --delete-secret-and-public-key "your@email.com"
```

5 Encryption & Decryption

5.1 Encrypting Messages

Encrypt data for secure transmission using various input methods:

```
# Encrypt single-line message using echo
echo "Secret message" | gpg --armor --encrypt -r recipient@email.com

# Encrypt from file
gpg --armor --encrypt -r recipient@email.com document.pdf

# Encrypt and save to specific output file
gpg --armor --encrypt -r recipient@email.com -o encrypted.asc plaintext.
    txt

# Encrypt for multiple recipients
gpg --armor --encrypt -r alice@example.com -r bob@example.com file.txt
```

Example

Encrypting Multi-line Content with Here Documents:

```
# Using here document (<<EOF syntax)
gpg --armor --encrypt -r recipient@email.com <<'EOF' > encrypted.asc
This is line 1 of my secret message.
This is line 2 with more content.
This is line 3 with final thoughts.
EOF

# Using here string (<<<) for single line
gpg --armor --encrypt -r recipient@email.com <<< "Quick secret message"

# Pipe from heredoc for complex content
cat <<'ENDMSG' | gpg --armor --encrypt -r recipient@email.com -o secret
    .asc
Subject: Confidential Information
Date: $(date)

This is a multi-line encrypted message
containing structured information.

```

```
Regards,
Your Name
ENDMSG
```

Pro Tip

Command Options Explained:

- `-armor` or `-a` – Produces ASCII-armored output (text format)
- `-encrypt` or `-e` – Performs encryption operation
- `-r recipient@email.com` – Specifies recipient's email/key ID
- `-o output.asc` – Specifies output filename
- `-local-user` – Selects which of your keys to use for signing

5.2 Decrypting Messages

Decrypt received data:

```
# Decrypt from file (output to terminal)
gpg --decrypt encrypted.asc

# Decrypt and save to file
gpg --decrypt encrypted.asc > decrypted.txt

# Decrypt with specific output filename
gpg --output decrypted.pdf --decrypt encrypted.pdf.gpg

# Decrypt directly from stdin
cat encrypted.asc | gpg --decrypt
```

Example

Decrypting Inline PGP Messages:

```
# Decrypt multi-line PGP message from heredoc
gpg --decrypt <<'PGPMSG'
-----BEGIN PGP MESSAGE-----

hQEMA1234567890ABCAQf9GvR...
[encrypted content here]
...xyz123==
-----END PGP MESSAGE-----
PGPMSG

# Or pipe from file
cat email_message.txt | gpg --decrypt
```

5.3 Symmetric Encryption

Encrypt with a passphrase (no key pair needed):

```
# Symmetric encryption (password-based)
gpg --symmetric --armor file.txt

# Symmetric with specific cipher algorithm
gpg --symmetric --cipher-algo AES256 --armor sensitive.zip

# Decrypt symmetric encryption
gpg --decrypt file.txt.asc

# Symmetric encryption from heredoc
gpg --symmetric --armor <<'EOF' > encrypted.asc
Sensitive data here
More sensitive content
EOF
```

Note

Symmetric vs Asymmetric:

- **Symmetric** – Uses the same passphrase for encryption/decryption. Simpler but requires secure passphrase sharing.
- **Asymmetric** – Uses public/private key pairs. More secure for multiple recipients but requires key management.

6 Digital Signatures

Digital signatures prove authenticity and integrity of data.

6.1 Creating Signatures

```
# Create detached signature (separate .sig file)
gpg --detach-sig document.pdf

# Create clearsigned text file (human-readable)
gpg --clearsign message.txt

# Sign and encrypt simultaneously
gpg --sign --encrypt -r recipient@email.com contract.docx

# Create inline signature
gpg --sign important-file.zip

# Sign message from heredoc
gpg --clearsign <<'EOF' > signed_message.asc
This is an important announcement.
It is signed to verify authenticity.
EOF
```

Note

Signature Types:
- **Detached** – Separate .sig file, most common for software distribution
- **Clearsign** – Human-readable text with signature wrapper
- **Inline** – Signature embedded in encrypted file
- **Sign & Encrypt** – Combined operation for authenticated encryption

6.2 Verifying Signatures

```
# Verify detached signature
gpg --verify document.pdf.sig document.pdf

# Verify clearsigned file
gpg --verify message.txt.asc

# Verify and extract content
gpg --output original.txt --decrypt signed.txt.gpg
```

Success/Tip

Signature Verification Results:
- **Good signature** – File has not been modified, signature matches identity
- **BAD signature** – File was tampered with or signature is invalid. Do not trust!

7 Git Integration

Sign your Git commits and tags with GPG for authenticity.

7.1 Configure Git for GPG

```
# Get your GPG key ID
gpg --list-secret-keys --keyid-format=long

# Configure Git to use GPG
git config --global user.signingkey YOUR_KEY_ID
git config --global commit.gpgsign true
git config --global tag.gpgsign true

# Set GPG program (if needed)
git config --global gpg.program gpg
```

7.2 Signing Commits & Tags

```
# Sign a commit (with auto-sign enabled)
git commit -m "Your commit message"

# Sign a commit (manual)
git commit -S -m "Signed commit message"

# Create signed tag
git tag -s v1.0.0 -m "Version 1.0.0"

# Verify signed commit
git log --show-signature

# Verify signed tag
git tag -v v1.0.0
```

Pro Tip

Why Sign Commits?

- **Authenticity** – Proves you authored the commit
- **Integrity** – Ensures code hasn't been modified
- **Non-repudiation** – Cannot deny creating the commit
- **Trust** – GitHub shows verified badge for signed commits

8 Advanced Key Operations

8.1 Edit Key Properties

Enter interactive key editing mode:

```
# Edit key interactively
gpg --edit-key your@email.com

# Within the interactive prompt, use:
# - expire       Change expiration date
# - adduid       Add additional user ID (email)
# - addkey       Add subkey
# - revkey       Revoke a key
# - trust        Change trust level
# - passwd       Change passphrase
# - save         Save changes and exit
# - quit         Exit without saving
```

8.2 Create Revocation Certificate

Warning

Create this immediately after generating your key!

```
# Generate revocation certificate
gpg --output revocation.crt --gen-revoke your@email.com

# Import revocation certificate (when needed)
gpg --import revocation.crt

# Publish revoked key to keyserver
gpg --keyserver hkps://keys.openpgp.org --send-keys YOUR_KEY_ID
```

Warning

Revocation Certificate Safety:

- Store in a separate, secure location (encrypted USB, password manager)
- Keep multiple copies in different physical locations
- Anyone with this certificate can revoke your key
- Use only if your private key is compromised or lost

9 Quick Reference

Key Management

```
gpg --gen-key
gpg -k
gpg -K
gpg --export
gpg --import
gpg --edit-key
gpg --gen-revoke
```

Signatures

```
gpg --sign
gpg --verify
gpg --clearsign
gpg --detach-sig
```

Encryption

```
gpg -e -r email
gpg -d file.gpg
gpg -c file
gpg --armor
```

Advanced

```
gpg --refresh-keys
gpgconf --kill
gpgconf --launch
gpg-connect-agent
```

9.1 File Extension Guide

Extension	Description
.gpg	Binary encrypted/signed file
.asc	ASCII-armored encrypted/signed file (text)
.sig	Detached signature file
.key	Exported key file (public or private)
.crt	Revocation certificate

10 Troubleshooting

10.1 Common Issues

Pro Tip

"Inappropriate ioctl for device" Error
Add to your shell's RC file (`.bashrc`, `.zshrc`, etc.):

```
export GPG_TTY=$(tty)
```

Then reload: `source ~/.bashrc` or restart your terminal.

Warning

"No secret key" Error
Possible causes:

- You don't have the private key for decryption
- Key was not properly imported
- Key has expired
- Wrong key selected

Solution:

```
# Verify you have the secret key
gpg -K

# Check key expiration
gpg --list-keys your@email.com
```

Pro Tip

GPG Agent Not Responding

```
# Kill and restart GPG agent
gpgconf --kill gpg-agent
gpgconf --launch gpg-agent

# Check agent status
gpg-connect-agent /bye
```

10.2 Useful Diagnostic Commands

```
# Show GPG version and capabilities
gpg --version

# Display detailed key information
gpg --list-keys --with-fingerprint your@email.com

# Show GPG configuration
gpgconf --list-components
```

```
# Test GPG functionality (encrypt/decrypt test)
echo "test" | gpg -e -r your@email.com | gpg -d

# Check keyserver connectivity
gpg --keyserver hkps://keys.openpgp.org --search-keys test@example.com
```

11 Best Practices & Security

Warning

Essential Security Measures:

1. **Strong Passphrases** – Minimum 20 characters, use password manager
2. **Key Rotation** – High-security: yearly; Standard: every 2-3 years
3. **Secure Backups** – Export private keys to encrypted storage in multiple locations
4. **Fingerprint Verification** – Always verify full 40-character fingerprint through separate channel
5. **Set Expiration Dates** – Force regular security review
6. **Generate Revocation Certificates** – Create immediately after key generation
7. **Never Share Private Keys** – Only distribute public keys

Pro Tip

Common Security Pitfalls to Avoid:

- Don't use weak or dictionary-based passphrases
- Don't store unencrypted private keys on cloud services
- Don't use keys without expiration dates
- Don't trust keys without verifying fingerprints
- Don't reuse the same key for everything (consider subkeys)

12 Additional Resources

12.1 Official Documentation

- **GNU Privacy Guard:** https://gnupg.org/
- **OpenPGP Standard:** https://www.openpgp.org/
- **Homebrew:** https://brew.sh/

12.2 Keyservers

- **keys.openpgp.org (Recommended):** https://keys.openpgp.org/
- **MIT PGP Keyserver:** https://pgp.mit.edu/
- **Ubuntu Keyserver:** https://keyserver.ubuntu.com/

12.3 Security Standards

- **AES** – Advanced Encryption Standard (symmetric)
- **RSA** – Rivest–Shamir–Adleman (asymmetric)
- **SHA-2** – Secure Hash Algorithm 2
- **ECC** – Elliptic Curve Cryptography

Security Reminder

Always verify key fingerprints through a separate channel.

Store your private keys securely and create backups.

Generate revocation certificates immediately after key creation.

Use GPG responsibly for authorized communications only.

Document Version: 2.0
Last Updated: November 5, 2025
Compiled with: LaTeX