PGP

Pretty Good Privacy and related things.

September 11, 2019

Overview

1. History
2. Terminology
3. Using GPG
4. Recent News
5. Cryptography
6. Ramble

History

1. PGP began as a program written by Phil Zimmermann in 1991 for symmetric encryption and decryption.
2. Zimmermann was prosecuted by the US for the illegal export of munitions. Started PGP Inc., which was later bought out.
3. Former PGP Inc. employees formed the PGP Corporation, which was later bought out.
4. PGP Inc. proposed a standard to the IETF called OpenPGP.
5. Two years later the FSF wrote GPG from OpenPGP.

Terminology

PGP:

- “Pretty Good Privacy” is a program by Phil Zimmermann.
- Able to encrypt, decrypt, sign, and verify things using many kinds of cryptography.

OpenPGP:

- Standardization of PGP for digital interchange.
  - Packets, ASCII armor, algorithm constants, fingerprints, keyrings, etc.
- Came from the PGP program written by Zimmermann.
- “Proposed standard” not a full standard, yet. RFC 4880.

GPG:

- “GNU Privacy Guard” is an implementation of OpenPGP as part of the GNU Project.

Using GPG: keys

GPG online manual.

Generating a key:	`$ gpg2 --full-gen-key`
Or, elliptic curves:	`$ gpg2 --full-gen-key --expert`
List public key:	`$ gpg2 --list-keys`
Export:	`$ gpg2 --armor --export {key}`
Receive key:	`$ gpg2 --recv-keys {key} --keyserver {URL}`
Send to keyserver:	`$ gpg2 --send-keys {key} --keyserver {URL}`

Using GPG: encrypting

Use hidden recipients! `--hidden-recipient` rather than `--recipient`

Encrypt file:

```
$ gpg2 --encrypt --hidden-recipient {key} \
       --output {output} {input}
```

Or, ASCII armor:

```
$ gpg2 --armor --encrypt ...
```

Decrypt file:

```
$ gpg2 --output {output} --decrypt {input}
```

Using GPG: signing

Sign and compress:	`$ gpg2 --output {output} --sign {input}`
Verify:	`$ gpg2 --verify {input}`
Verify and decompress:	`$ gpg2 --output {output} --decrypt {input}`
Clearsign:	`$ gpg2 --output {output} --clearsign {input}`
Detached signatures:	`$ gpg2 --output {output} --detach-sig {input}`

Recent News: SKS Keyserver Poisoning

Full write-up.

- Keyservers are a registry of PGP keys.
- OpenPGP allows key signing: submitting a signature of someone's public key establishing your trust in the owner of that key. Sometimes done through "key signing parties." This is essential to the web-of-trust model.
- SKS keyservers are a popular, high-performing, distributed pool of keyservers. SKS keyservers, and many other kinds of keyservers, do not stop random people from signing your key.

Recent News: SKS Keyserver Poisoning

The vulnerability is from spamming a key with signatures (order of 150,000). This breaks GPG, but is isn't a vulnerability in the OpenPGP specification itself.

Possible solutions:

- Modify the OpenPGP spec to require approval of key signatures before they attach themselves to a key.
- Modify keyservers to require some kind of verification or authentication for sending your key, or sending your key signatures.

Cryptography

Symmetric cryptography:

- Cryptographic algorithms that use the same key for encryption and decryption.
- Key should be kept secret.

Asymmetric cryptography:

- Cryptographic algorithms that use different keys for encryption and decryption.
- One key can be shared, one key should be kept secret.

Digital signatures:

- Use your private key to sign data, which can be verified through your public key.
- Provides integrity, authentication, and non-repudiation.

Cryptography

RSA:

- One such asymmetric cryptographic algorithm.
- Let e, d, n be large positive numbers such that $\forall m$ where $0 \leqslant m \leqslant n$, $(m^e)^d \equiv m \mod n$.
- ... and lots more math.

DSA:

- Like RSA, but faster for signing and decrypting, but slower for verifying and encrypting.

ECC:

- Elliptic-curve cryptography
- Smaller keys for equal strength crypto
- Be careful which curves you use
  - Prefer: EdDSA / Ed25519 / Curve25519
  - Avoid: EcDSA (NSA backdoor)
- Weaker to quantum cryptography than RSA.

Ramble

Ramble provides transparent and secure PGP-based messaging.

- Users introduce themselves to the server.
- Send messages as part of conversations.
- View messages.

All communication is done with your own PGP keys.

- Server requires key verification through nonce signatures before communicating.
- All messages are encrypted, the server cannot know their contents nor their recipients.
- **Even if a third party watches all connections they will gain no useful knowledge.**[1]

https://github.com/esote/ramble

[1] Assuming you use hidden recipients for encrypted messages.