PGP, GPG and who the hell is Alice?

Zsolt Hegyi

December 6, 2015

(In)secure mail

E-mail is insecure by design

(In)secure mail

E-mail is insecure by design

What can we do?

Mitigations?

- Secure the servers?

Mitigations?

- Secure the servers? ✗
  The NSA is a pretty worthy opponent... Better assume that all infrastructure is under their control.
- Secure the clients?

Mitigations?

- Secure the servers? ✗
  The NSA is a pretty worthy opponent... Better assume that all infrastructure is under their control.
- Secure the clients? ✗
  Everyone would have to use the same mail client, which doesn't happen.
- Secure the content?

Mitigations?

- Secure the servers? ✗
  The NSA is a pretty worthy opponent... Better assume that all infrastructure is under their control.
- Secure the clients? ✗
  Everyone would have to use the same mail client, which doesn't happen.
- Secure the content? ✓
  Just encrypt it!

PGP

Phil Zimmerman, 1991

PGP

Phil Zimmerman, 1991

US Export regulations: Crypto = munitions

PGP

Phil Zimmerman, 1991

US Export regulations: Crypto = munitions

Solution: First Amendment the shit out of it (12 books, 6000 pages)

GPG – GNU Privacy Guard

GPG – GNU Privacy Guard

Based on PGP.

GPG – GNU Privacy Guard

Based on PGP.

Comes default with most Linux distros.

GPG – GNU Privacy Guard

Based on PGP.

Comes default with most Linux distros.

FOSS.

GPG – GNU Privacy Guard

Based on PGP.

Comes default with most Linux distros.

FOSS.

Cool.

Problem #1: Did the e-mail really come from the sender?

Problem #1: Did the e-mail really come from the sender?

Pretty easy to fake an e-mail

World War 3?

```
EHLO
MAIL FROM: <obama@thewhitehouse.gov>
RCPT TO: <putin@zeduma.ru>
DATA
Bombs away, punk!
.
```

Problem #1: Did the e-mail really come from the sender?

Pretty easy to fake an e-mail

World War 3?

```
EHLO
MAIL FROM: <obama@thewhitehouse.gov>
RCPT TO: <putin@zeduma.ru>
DATA
Bombs away, punk!
.
```

Solution: Signing!

Creating a keypair

Creating a keypair

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

Creating a keypair

Creating a keypair

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

Options

- Key type: RSA & RSA
- Key size: 4096 bit (or bigger)
- Key expiration: 1 year recommended
- ALWAYS use a keyphrase

Creating a keypair

Creating a keypair

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

Options

- Key type: RSA & RSA
- Key size: 4096 bit (or bigger)
- Key expiration: 1 year recommended
- ALWAYS use a keyphrase

This is the **master keypair.**

Helping the entropy pool build

```
dd if=/dev/sda of=/dev/zero
```

Helping the entropy pool build

```
dd if=/dev/sda of=/dev/zero
```

Question time (while the keys generate)! Topics:

- History of crypto
- GPG
- Keys

Exporting our key

Exporting the private key to *secret.pgp*

```
gpg --output secret.pgp
--export-secret-keys --armor KEYID
```

Importing a private key from *secret.pgp*

```
gpg --allow-secret-key-import --import
secret.pgp
gpg --import public.gpg
```

KEYID can be the e-mail address associated, or the last 64 bits of the public key fingerprint.

Important note: Usage of 32 bit fingerprints is discouraged. Can be bruteforced.

Signing

Signing a *message*

gpg --clearsign message

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Hey cousin, let's go to bowling!

Roman
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iQI...=YQJD
-----END PGP SIGNATURE-----
```

Signing

Signing a *message*

gpg --clearsign message

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Hey cousin, let's go to bowling!

Roman
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iQI...=YQJD
-----END PGP SIGNATURE-----
```

Verifying a signature

Verification

```
gpg --verify message.asc
```

```
Signature made 2015. dec.  6., vasárnap, 14.06.30 CET
using RSA key ID 4C2037B1
Good signature from "Zsolt Hegyi <hegyizs@crysys.hu>"
```

Problem #2a: How do we get our public keys to the other side?

Problem #2a: How do we get our public keys to the other side?

NSA can just strip the signature, and insert theirs, along with the modified content.

Problem #2a: How do we get our public keys to the other side?

NSA can just strip the signature, and insert theirs, along with the modified content.

Problem #2b: How do we make sure we are talking to the right person?

Problem #2a: How do we get our public keys to the other side?

NSA can just strip the signature, and insert theirs, along with the modified content.

Problem #2b: How do we make sure we are talking to the right person?

NSA can just generate keys in our name, and do the previous!

Problem #2a: How do we get our public keys to the other side?

NSA can just strip the signature, and insert theirs, along with the modified content.

Problem #2b: How do we make sure we are talking to the right person?

NSA can just generate keys in our name, and do the previous!

Solution: Signing!

Problem #2a: How do we get our public keys to the other side?

NSA can just strip the signature, and insert theirs, along with the modified content.

Problem #2b: How do we make sure we are talking to the right person?

NSA can just generate keys in our name, and do the previous!

Solution: Signing! ... what now?

Getting keys to the other side

Solution: Upload it to your own server...

Importing & Exporting public keys

Exporting the public key to *public.pgp*

```
gpg --output public.pgp --armor --export KEYID
```

Importing & Exporting public keys

Exporting the public key to *public.pgp*

```
gpg --output public.pgp --armor --export KEYID
```

Importing a public key from *public.pgp*

```
gpg --import public.gpg
```

Getting keys to the other side

Solution: Upload it to your own server...

Getting keys to the other side

Solution: Upload it to your own server... Or just use a keyserver.

Keyservers

Working with keyservers

Downloading a key:

`gpg --keyserver` *`keyserver`* `--search KEYID`

Uploading a key:

`gpg --keyserver` *`keyserver`* `--send-keys KEYID`

Verifying identities

To verify a key, you must first meet... A good way to get your key signed is by:

- Attending conferences and key signing / cryptoparties
- Working on FOSS projects
- Going to your local Hackerspace
- Asking your friends

Verifying identities

To verify a key, you must first meet... A good way to get your key signed is by:

- Attending conferences and key signing / cryptoparties
- Working on FOSS projects
- Going to your local Hackerspace
- Asking your friends

Preparation protips:

- `gpg-key2ps` or DuckDuckGo `pgp key slip generator`
- Proof of identity (passport, ID, drivers license)
- Someone who can vouch for you

Fingerprinting

PGP fingerprints are hashes of public keys.

Fingerprinting

PGP fingerprints are hashes of public keys.

Getting fingerprint of **KEYID**

```
gpg --fingerprint KEYID
```

Signing keys

After a thorough identity check, it's time to sign the other party's key. But first, verify that the **full** fingerprint matches the other party's fingerprint. Also, you might want to check the signatures on the key before.

Checking key signatures

```
gpg --check-sigs KEYID
```

Signing keys

After a thorough identity check, it's time to sign the other party's key. But first, verify that the **full** fingerprint matches the other party's fingerprint. Also, you might want to check the signatures on the key before.

Checking key signatures

```
gpg --check-sigs KEYID
```

Signing a key

```
gpg --sign-keys KEYID
```

Signing keys

After a thorough identity check, it's time to sign the other party's key. But first, verify that the **full** fingerprint matches the other party's fingerprint. Also, you might want to check the signatures on the key before.

Checking key signatures

```
gpg --check-sigs KEYID
```

Signing a key

```
gpg --sign-keys KEYID
```

After being done with this, just upload it back to the

Trust issues

When can we say that a key is trusted?

Trust issues

When can we say that a key is trusted?

Set rules for trusting others and signed materials:

- Trust is on a 1–5 scale, from ”Untrusted” to ”Ultimately trusted”

Trust issues

When can we say that a key is trusted?

Set rules for trusting others and signed materials:

- Trust is on a 1–5 scale, from ”Untrusted” to ”Ultimately trusted”
- Own key is automatically ”Ultimately trusted”

Trust issues

When can we say that a key is trusted?

Set rules for trusting others and signed materials:

- Trust is on a 1-5 scale, from ”Untrusted” to ”Ultimately trusted”
- Own key is automatically ”Ultimately trusted”
- Can set how much we trust a key when we sign it

Trust issues

When can we say that a key is trusted?

Set rules for trusting others and signed materials:

- Trust is on a 1-5 scale, from ”Untrusted” to ”Ultimately trusted”
- Own key is automatically ”Ultimately trusted”
- Can set how much we trust a key when we sign it
- If a key is signed by a ”trusted” key, it is considered trusted, but less trusted (marginal trust).

Trust issues

When can we say that a key is trusted?

Set rules for trusting others and signed materials:

- Trust is on a 1-5 scale, from ”Untrusted” to ”Ultimately trusted”
- Own key is automatically ”Ultimately trusted”
- Can set how much we trust a key when we sign it
- If a key is signed by a ”trusted” key, it is considered trusted, but less trusted (marginal trust).
- Default for verification: 1 completely trusted signature, or 3 marginally trusted signatures.

Trust issues

When can we say that a key is trusted?

Set rules for trusting others and signed materials:

- Trust is on a 1-5 scale, from ”Untrusted” to ”Ultimately trusted”
- Own key is automatically ”Ultimately trusted”
- Can set how much we trust a key when we sign it
- If a key is signed by a ”trusted” key, it is considered trusted, but less trusted (marginal trust).
- Default for verification: 1 completely trusted signature, or 3 marginally trusted signatures.

Web of Trust

Key management

Simple!

- All keys are stored in the local keystore.
- Importing a key puts it into the local keystore.
- The key store also contains a ”trust database”.

Listing key store

```
gpg --list-keys
```

Choosing default key

```
gpg --default-key KEYID
```

The PGP cryptosystem

Not the topic of this presentation, brief explanation:

The PGP cryptosystem

Not the topic of this presentation, brief explanation:

- **SYMMETRIC CRYPTO:** One-time session key is generated, plaintext is encrypted with it.

The PGP cryptosystem

Not the topic of this presentation, brief explanation:

- **SYMMETRIC CRYPTO:** One-time session key is generated, plaintext is encrypted with it.
- **ASYMMETRIC CRYPTO, PUBLIC:** Session-key is encrypted with *recipient's* public key. It is bundled with the encrypted data, giving the message.

The PGP cryptosystem

Not the topic of this presentation, brief explanation:

- **SYMMETRIC CRYPTO:** One-time session key is generated, plaintext is encrypted with it.
- **ASYMMETRIC CRYPTO, PUBLIC:** Session-key is encrypted with *recipient's* public key. It is bundled with the encrypted data, giving the message.
- **ASYMMETRIC CRYPTO, PRIVATE:** The message is digitally signed using the *sender's* private key.

Encryption and decryption

Encrypting *message*

```
gpg --output message.gpg --encrypt
--armor --recipient RECPT_KEYID message
```

Shorthand for -encrypt: -e

Encryption and decryption

Encrypting *message*

```
gpg --output message.gpg --encrypt
--armor --recipient RECPT_KEYID message
```

Shorthand for -encrypt: -e

Decrypting *message*

```
gpg --output message --decrypt
message.gpg
```

Shorthand for -decrypt: -d

Encryption and decryption

Encrypting *message*

```
gpg --output message.gpg --encrypt
--armor --recipient RECPT_KEYID message
```

Shorthand for -encrypt: -e

Decrypting *message*

```
gpg --output message --decrypt
message.gpg
```

Shorthand for -decrypt: -d

And one more thing: Shorthand for ASCII armoring: -a

Problem #3: Making this usable in an everyday setting.

Problem #3: Making this usable in an everyday setting.

Solutions: Platform-independent:

- Thunderbird + Enigmail

Windows

Problem #3: Making this usable in an everyday setting.

Solutions: Platform-independent:

- Thunderbird + Enigmail

Windows (you're already pwned):

Problem #3: Making this usable in an everyday setting.

Solutions: Platform-independent:

- Thunderbird + Enigmail

Windows (you're already pwned):

- gpg4win
  Contains the actual gpg binary, along with GNU Privacy Assistant

Problem #3: Making this usable in an everyday setting.

Solutions: Platform-independent:

- Thunderbird + Enigmail

Windows (you're already pwned):

- gpg4win
  Contains the actual gpg binary, along with GNU Privacy Assistant

*nix:

- GNU Privacy Assistant
- Keychain application that comes with your distro

OpSec hints&tips:

- Metadata $\rightarrow$ bad
  Recipient and subject fields are right out in the open!

OpSec hints&tips:

- Metadata $\rightarrow$ bad
  Recipient and subject fields are right out in the open! ”Master decryption password” totally doesn’t attract prying eyes!

OpSec hints&tips:

- Metadata $\rightarrow$ bad
  Recipient and subject fields are right out in the open! ”Master decryption password” totally doesn’t attract prying eyes!
- Use a password manager!
  - KeePassX is my personal favorite
  - Using ”cloud-based” stuff is like sticking your dick into random holes. Don’t.

OpSec hints&tips:

- Metadata $\rightarrow$ bad
  Recipient and subject fields are right out in the open! ”Master decryption password” totally doesn’t attract prying eyes!
- Use a password manager!
  - KeePassX is my personal favorite
  - Using ”cloud-based” stuff is like sticking your dick into random holes. Don’t.
- **Do not** trust people by default!

OpSec hints&tips:

- Metadata $\rightarrow$ bad
  Recipient and subject fields are right out in the open! ”Master decryption password” totally doesn’t attract prying eyes!
- Use a password manager!
  - KeePassX is my personal favorite
  - Using ”cloud-based” stuff is like sticking your dick into random holes. Don’t.
- **Do not** trust people by default!
- **Always** assume an almost-omnipotent adversary!

OpSec hints&tips:

- Metadata $\rightarrow$ bad
  Recipient and subject fields are right out in the open! ”Master decryption password” totally doesn’t attract prying eyes!
- Use a password manager!
  - KeePassX is my personal favorite
  - Using ”cloud-based” stuff is like sticking your dick into random holes. Don’t.
- **Do not** trust people by default!
- **Always** assume an almost-omnipotent adversary!
- **Always** sign your mails, and always encrypt when the other party is also using PGP!

OpSec hints&tips:

- Metadata $\rightarrow$ bad
  Recipient and subject fields are right out in the open! ”Master decryption password” totally doesn’t attract prying eyes!
- Use a password manager!
  - KeePassX is my personal favorite
  - Using ”cloud-based” stuff is like sticking your dick into random holes. Don’t.
- **Do not** trust people by default!
- **Always** assume an almost-omnipotent adversary!
- **Always** sign your mails, and always encrypt when the other party is also using PGP!
  One single encrypted email amongst all the others is always suspicious...

The end goal: an end-to-end encrypted world...

The end goal: an end-to-end encrypted world... and really mad intelligence services

Questions?

Zsolt Hegyi
<hegyizs@crysys.hu>
C444 DEBC 46D0 F62D 1324
3019 2D05 8DCF 4C20 37B1