Functional purposes of cryptography

- Confidentiality
  - ensuring illegibility to outsiders

- Authentication
  - ensuring ostensible and actual sender are one and the same

- Data integrity
  - ensuring non-alteration in transit
GPG – GNU Privacy Guard

- An implementation of OpenPGP

“Open-PGP software uses a combination of strong public-key and symmetric cryptography to provide security services for electronic communications and data storage. These services include confidentiality, key management, authentication, and digital signatures.”

Abstract, OpenPGP RFC4880

Cryptographic processing

<table>
<thead>
<tr>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>plaintext</td>
<td>cryptogram</td>
</tr>
<tr>
<td>cipher</td>
<td>reverse cipher</td>
</tr>
<tr>
<td>cryptogram</td>
<td>plaintext</td>
</tr>
</tbody>
</table>
Cryptographic strength

- Difficulty of recovering plaintext from ciphertext without the key
- Does not depend on the cipher (algorithm)
  - standard algorithms are published
- So everything depends on the key
  - key loss/compromise/interception=Achilles heel

Key distribution management

- Good: protect it
- Better: don’t distribute it
Best key distribution philosophy

- DON’T
- “What good would it do after all to develop impenetrable cryptosystems, if their users were forced to share their keys with a Key Distribution Center that could be compromised by either burglary or subpoena?” — Whitfield Diffie

2 broad categories

- Secret-key cryptography
  versus
- Public-key cryptography (1970)
Known synonymously as:

- One technology
  - single-key
  - private-key
  - symmetric
  - secret-key
  - shared-key
  - conventional

- Versus the other
  - dual-key
  - public-key
  - asymmetric

Decrypt-key distribution demands

- Secret-key
  - distribute but protect
  - low risk

- Public-key
  - none, you don’t need to distribute it
  - no risk
Keys: secret-key crypto

Encryption (data sender)

plaintext

cipher

ciphertext

ciphertext

Decryption (data receiver)

plaintext

reverse ciphertext

(ciphertext)

Keys: public-key crypto

Encryption

plaintext

ciphertext

ciphertext

Decryption

plaintext

reverse ciphertext

(ciphertext)

(different key)
Wait a minute…

- who sends the key(s) to whom?
- what/which key(s) does he send?

Secret-key crypto: data sender is key sender

Key sender

Encryption (data sender)
- plaintext
- cipher
- cryptogram

Decryption (data receiver)
- cryptogram
- reverse cipher
- plaintext

Key sent (same key)
What if it’s public-key? more guys/keys

- if there are 2 guys, there are 2 key pairs (4 keys)
- who sends the key?
- what key does he send?
- what does that accomplish?

Well…

- only public keys can be sent!
- so either guy could be the key sender
- and he would send his public key (only! ever!)
- depending who sends, accomplishes
  - confidentiality, or
  - authentication
Data receiver as key sender

Encryption (data sender)

plaintext

cipher

cryptogram

Key sent (data receiver’s public key)

Decryption (data receiver)

cryptogram

inverse cipher

plaintext

Key sender (data receiver’s private key)

Functional achievement checklist

Data receiver as key sender

- Confidentiality ✓
- Authentication
- Data integrity
Data sender as key sender

Functional achievement checklist

Data sender as key sender

- Confidentiality
- Authentication ✓
- Data integrity
But can’t we have both together?

- Confidentiality ✓
- Authentication ✓
- Data integrity

Certainly! if you just encrypt and decrypt twice

Encrypting the whole message twice is too expensive!!

- Make a little token\(^1\) from a big message with a hash function\(^2\)
- Encrypt the token instead of the message

\(^1\)also called a message digest or hash
\(^2\)also called a digest function, like MD5 or SHA1 or RIPEMD-160
Confidential and authentic*

Encryption (data sender)
plaintext
\[ \text{cipher} \]
\[ \text{cryptogram} \]

Decryption (data receiver)
cryptogram
\[ \text{reverse cipher} \]
\[ \text{plaintext} \]

H - hash
S - signature

authentic but not confidential*

Encryption (data sender)
plaintext
\[ \text{cipher} \]
\[ \text{cryptogram} \]

Decryption (data receiver)
plaintext
\[ \text{S} \]
\[ \text{H} \]

H - hash
S - signature

* gpg encrypt and sign

* gpg sign only, also useful

© David Morgan 2006-2015
gpg default directories and files

```
/  
   |   
  home  
   |   
  tom   dick   harry  
   |       |       |       
  .gnupg  .gnupg  .gnupg  
   |       |       |       
  gpg.conf pubring secring  gpg.conf pubring secring  gpg.conf pubring secring  
   |       |       |       
  gpg.conf pubring secring  gpg.conf pubring secring  gpg.conf pubring secring  
```

---

gpg key management options

- `--gen-key`
  - create keypair and store on disk
- `--export`
  - take public key from disk and output to file
- `--import`
  - take public key from file and output to disk
**gpg encryption options**

- `--encrypt <file> --recipient <user>`
  - encrypt file with user’s pubkey from disk
- `--decrypt <file>`
  - decrypt file using private key from disk that matches public key with which file was encrypted

**gpg signing options**

- `--sign <file>`
  - create digest of file, encrypt it with private key
- `--verify`
  - decrypt sender’s digest, generate your own, check they’re the same
Example: believing in fedora

1) this file’s digests, for the other files, make them believable

2) signature on digests’ file makes it believable

Get fedora project’s public key
Add fedora’s key to your keyring

```
[root@CHANG ~]# gpg --import Desktop/pubkey.asc
gpg: key 4F2A6F02, public key "Fedora Project <fedora@redhat.com>" imported
gpg: Total number processed: 1
  imported: 1
  3 marginal(s) needed, 1 complete(s) needed, PGP trust model
  depth: 0  valid: 1  trust: 0*, 0q, 0m, 0f, ilu
[root@CHANG ~]#
[root@CHANG ~]# gpg --list-keys
/root/.gnupg/pubring.gpg

pub 1024D/B452A986 2006-06-04
uid Daniel Moore (keep it private) <dmoores@dslextreme.com>
sub 2048g/D6D66EFC 2006-06-04
pub 1024D/1D2EF028 2006-06-04
uid David Morgan (teach linux) <dmorgan@dslextreme.com>
sub 2048g/40570861 2006-06-04
pub 1024D/4F2A6F02 2003-10-27
uid Fedora Project <fedora@redhat.com>
sub 1024g/FA93E34 2003-10-27
```

Use it: file really from fedora?

```
[root@CHANG ~]# gpg --verify SHA1SUM
  gpg: Signature made Tue Mar 14 21:38:22 2006 PST using DSA key ID 4F2A6F02
  gpg: Good signature from "Fedora Project <fedora@redhat.com>"
  gpg: WARNING: This key is not certified with a trusted signature!
  gpg: Public key fingerprint: CA64 4B99 6F27 744E 8612 7CDE B442 69D0 4F2A 6F02
[root@CHANG ~]#
```

...if the key is really fedora’s, the file is really from them
we believe so
Do downloads check out?

```
Do downloads check out?
```

<table>
<thead>
<tr>
<th>Session</th>
<th>Edit</th>
<th>View</th>
<th>Bookmarks</th>
<th>Settings</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>{root@CHANG FC3files}# cat SHA1SUM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---BEGIN PGP SIGNED MESSAGE-----

Hash: SHA1

cdeb3303f8da1ff521c8d084d&7275f0e6a4d8f
35f460e0b3f1c9d6b04f2f10a9f500da2c50e400
2c06f869a092f259af2c866527c6f1386-1386-iso
58308dfc3c4f2722979e82d12d350af5c080a707
07f77f8406905cf7776a-1386-disc4.iso
366b0d20f62bc5e4c2eadeb13334b79bd52e9-1386-disc5.iso
---BEGIN PGP SIGNED MESSAGE-----

Version: GnuPG v1.2.1 (GNU/Linux)

[1DDBBB8EFb0E79b99bf1AD7Y5A2H6X06Z2vN3E5xtgLa4U/0gEgE0Fp]

-3XK833K38L/11F3HJv3fA69=

-----BEGIN PGP SIGNATURE-----

Do downloads check out?

OK, except what’s up with disc2??

actual

ostensible

per fedora and we believe it!

---BEGIN PGP SIGNATURE-----

| {root@CHANG FC3files}# sha1sum FC-5 |
| {root@CHANG FC3files}# sha1sum FC-5 |

```
gpa – GUI frontend to gpg
```

```
OK, except what’s up with disc2??
```
Enigmail – integrates GPG+email

Others:  http://www.gnupg.org/related_software/frontends.html

Info

- official page
  - http://www.gnupg.org
- GPG Mini HowTo
  - good, quick bare essentials
- GNU Privacy Handbook
  - more thorough and explanatory
- RFC4880 (OpenPGP message format)