Public Key Infrastructures

Chapter 5
Trust Models

Cryptography and Computer Algebra
Vangelis Karatsiolis
Alexander Wiesmaier

We have seen certificates.

Why do we trust in public keys?

Example

Authenticated by digital signature
Click on icon

The browser is shipped with trusted authorities

Click on view

Built-in object token
Direct trust

Public Key built-in.

User can check fingerprint.

Fingerprint

It is the hash value of the certificate

Example:

*Calculate hash value* (e.g. SHA1)

Fingerprint calculation

The key owner transmits/publishes the fingerprint

by phone
web page
newspaper or public document
CD-ROM
Phone verification

Web page verification

Federal Gazette

BNetzA publishes the public key (still RegTP in 2005)

CD-ROM verification

~# gpg --list-public-keys /root/.gnupg/pubring.gpg

pub 2048R/3D25D3D9 1999-03-06 SuSE Security Team <security@suse.de>
pub 1024/D/9C800ACA 2000-10-19 SuSE Package Signing Key <build@suse.de>
sub 2048g/8495160C 2000-10-19 [expires: 2006-02-12]
Direct trust

A user obtains the public key directly from owner or has the fingerprint of the public key verified directly by the owner.

Examples of direct trust

Certificate installation

Web page
Installed certificate

Original CD-ROM
Every customer obtains original-CDs that contain the public key. The medium offers protection against manipulation. Key compromise as well as CD forgery can become known through the media.

```
~# gpg --list-public-keys
/root/.gnupg/pubring.gpg
------------------------pub  2048R/3D25D3D9 1999-03-06 SuSE Security Team
<security@suse.de>
pub  1024D/9C800ACA 2000-10-19 SuSE Package Signing Key
<build@suse.de>
sub  2048g/8495160C 2000-10-19 [expires: 2006-02-12]
```

Trust in key = Trust in original-CDs

Direct trust: example

Signed RPMs
RedHat Package Manager
Software installation in Linux
Source: miscellaneous URLs
Is a URL trusted?
(e.g. is ftp://ftp.suse.com/.../openssl-0.9.6g-30.i586.rpm one malicious version of OpenSSL?)
Solution: The Linux-Distributor signs the RPM-Packet with GPG (GNU Privacy Guard)
Public Key inside the original-CD for verification
```
~# rpm --checksig ./openssl-0.9.6g-30.i586.rpm
openssl-0.9.6g-30.i586.rpm: md5 gpg in Ordnung
~#
```

Direct trust: example (SSH)

Example 3: SSH
SSH: Secure Shell
Cryptographically secured alternative for telnet and ftp.
User authentication based on password or public key.
Computer authentication based always on public key.

Public key based user authentication:
The user creates a key pair on the client side.
The user deposits the public key on the server.
The user must remove compromised keys from the host.
Direct trust: example (SSH)

Public Key based computer authentication:
- Protection against DNS- or IP-Spoofing.
- The public key of the server is transmitted during the first login.
- The user must verify the fingerprint by asking the server’s administrator.
- Warning in case of public key alteration.

```
-> ssh cdcnt56
The authenticity of host 'cdcnt56 (130.83.23.156)' can’t be established.
Are you sure you want to continue connecting (yes/no)? yes
19752: Warning: Permanently added 'cdcnt56,130.83.23.156' (RSA) to the list of known hosts.
```

Direct trust: summary

It is the basis for all other trust models
- One knows which keys are authentic and why they are considered authentic
- Bad scalability: in pairs, authentic key exchange of all keys requires (e.g. by verifying the fingerprint) \( n \times (n-1) / 2 \) verifications.

Web of trust

Used in PGP.

Every user has a public keyring that contains:
- Name
- Public key
- Key legitimacy (or validity)
- Owner trust (or trust)
- Signature(s)
Public keyring

Name
Key legitimacy
Owner trust
Signatures

Alice’s public keyring

Key Legitimacy

Three levels

unknown
Nothing is known about this key.
marginal
The key probably belongs to the name.
complete
The key definitely belongs to the name.

Owner trust

Five levels

unknown
Nothing is known about the user with this name.
none
The user with this name does not verify or verifies wrongly a public key before signing it.
marginal
The user with this name verifies a public key partially before signing it.
complete
The user with this name verifies a public key thoroughly before signing it.
ultimate
If the user with this name is the user himself.
Direct trust

Verification of the fingerprint directly with the owner of the key.

Then the user signs the key.

Key legitimacy is "complete".

PGP fingerprint (v4)

SHA-1 hash of this input:

Packet Tag
Length
Version
Creation Time
Public Key Algorithm
Public Key (RSA case)

Example

Bob’s key legitimacy is complete for Alice because she has signed the key by verifying the fingerprint.
Key legitimacy computation

In PGP, the users sign the public key of other users.

The user assigns owner trust to names.

The key legitimacy is computed from the owner trust and only for "complete" valid keys.

Complete key legitimacy

If the key is signed by the user himself.
Complete key legitimacy

If the key is signed by at least one name with owner trust complete.

Marginal key legitimacy

If the key is signed by a number of names, with owner trust marginal, less than x.

PGP newer versions

Key’s legitimacy is either verified or not. Although one marginal, the key is verified.
Older versions

Configurable trust model

Trusted Introducers

Alice uses owner trust values of Bob.

Chaining trusted introducers

Alice uses trusted introducers of Bob.

The Web of Trust

From PGP: Pretty Good Privacy by Simon Garfinkel
Using PGP...

What is PGP doing?

Encrypt Message
Encrypt Message (multiple recipients)
Sign Message
Sign and then encrypt
Symmetric encryption
Decrypt
Check Signatures

Key Management

Key Generation
Add Key to Key Ring
Remove Key From Key Ring
Extract Key From Key Ring
View Contents of Key Ring
Revoke Public Key

Enigmail
Enigmail

GPG Program

OpenPGP implementation

Key Creation

gpg --gen-key

Alice Crypto <alice@informatik.de>

Key List

gpg --list-keys

list keys
Revocation Certificate

gpg --output revoke.asc --gen-revoke keyid

revoke key

Keep this certificate secret
others could revoke the key

Key Export

gpg --output alice.gpg --export keyid

export key

Option: --armored (ASCII and not binary)

Key Import

gpg --import alice.gpg

import key

Edit Key

gpg --edit-key keyid

edit key

--fingerprint (fpr)
--sign
--check
Key Deletion

```
gpg --delete-secret-key keyid (private)
gpg --delete-key keyid (public)
```

*delete key*

Encryption

```
gpg -a -e -r keyid <filename>
```

*encrypt*
- `a`: armored (ASCII)
- `e`: encrypt
- `r`: UID

Decryption

```
gpg -d <filename>
```

*decrypt*
- `o`: output filename

Signature

```
gpg -a -u uid -s <filename>
```

*sign*
Clear Signature

gpg -a -u uid --clearsign <filename>

clear sign

Detached Signature

gpg -a -u uid --detach-sign <filename>
detach sign

Verify Signature

gpg --output <filename> --decrypt <filename>
verify signature

Verify Detached-Clear Signature

gpg --verify <filename>
verify detached signature
Scenario

![Diagram of a scenario with characters and arrows connecting them.]

Hierarchical trust

- Certification Authority (CA)
  - trust anchor
  - issues certificates

Alice

Bob

Carl

Why does Alice trust in Doris' key?

Alice sets trust to Bob's key “complete”

Hierarchical trust

- root CA
  - DFN PCA

- TUD CA
  - TUD Student CA
  - TUD Employee CA
  - Alice
  - Bob
  - Carl
  - Doris

- Uni Gießen

- Emil

Why does Alice trust in Doris' key?
Alice trusts in Doris' public key because:

Doris' certificate is the last certificate in path.
she trusts the anchor directly.

Hierarchical trust

Emil to Alice

Trusted List

Every participant has a list of trusted CAs.
Alice trusts TC$_2$ und TC$_3$
Every user maintains an own list (like in the Web of Trust)
Trusted List: certification path

Alice - Bob - Carl - Doris - Emil - Fred - Gerd - Hans

Alice to Fred

Common Root

Alice - Bob - Carl - Doris - Emil - Fred - Gerd - Hans

Every user who trusts TC1, accepts every other end-user certificate.
Two hierarchies

Common root

Common Root: certification path

Cross-certification

TC2 issues one CA-certificate for TC3.
TC3 issues one CA-certificate for TC2.
⇒ Every user who trusts TC3, accepts every certificate, that was issued by TC2 (or a subordinate CA).
⇒ Every user who trusts TC2, accepts every certificate, that was issued by TC3 (or a subordinate CA).

Not always bilateral
Cross-certification: certification path

TC2 issues one CA-certificate to TC7 and vice versa.
⇒ Hans accepts the certificate of Emil and vice versa.
⇒ Emil does not accept the certificate of Fred.

Another possibility:

TC4 issues one CA-certificate to TC6 and vice versa.
⇒ Alice accepts the certificate of Fred and vice versa.
⇒ Fred does not accept the certificate of Emil.
Cross-certification

Alice to Fred

Bridge

Idea: Bridge TC has cross-certifications with TC2 and TC3.
⇒ Alice accepts all certificates beneath TC3.
⇒ Fred accepts all certificates beneath TC2.

Bridge: certification path

Bridge Trust Center

The bridge TC acts as a bridge.
This TC is not subordinate to a third CA.

Interesting for corporate CAs that:

want to enable secure communication for their users outside the organisation's borders.
do not want to be subordinate to a third CA.

Most known project: European Bridge-CA
URL: www.bridge-ca.org
European Bridge-CA: Participants

Project Management:

founder members (in 2000):

other members

 Relevant extensions

Basic Constraints

Identifies whether the subject of the certificate is a CA
the maximum number of non-self-issued intermediate certificates that may follow this certificate in a valid certification path.

It is marked critical
If pathlength is not present => no limit

BasicConstraints ::= SEQUENCE {
  ca BOOLEAN DEFAULT FALSE,
  pathLenConstraint INTEGER (0..MAX) OPTIONAL }

../Certificates/text/CSCA_BasicConstraints.cxt (text)
../Certificates/Country_Signing_CA.cer (bin)

Basic Constraints (2)

Root CA

Subordinate CA

Alice

ca:true

ca:true

ca:false