PGP: Pretty Good Privacy

Ibéria Medeiros

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa

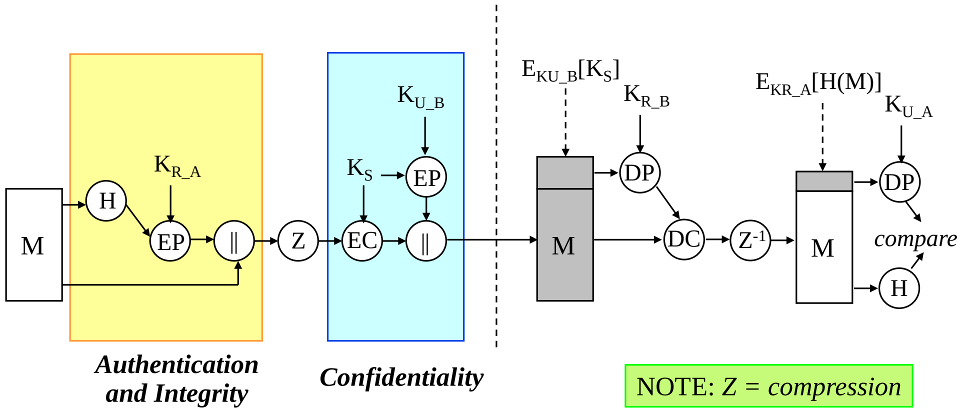
PGP - *Pretty Good Privacy*

- PGP offers confidentiality, integrity and authenticity for email or file storage services
- Some reasons for the success of PGP
  - selected the best available cryptographic algorithms
    - » digital signatures: DSS/SHA and RSA/SHA
    - » encryption: CAST-128, IDEA or 3DES *with* ElGamal or RSA
    - » compression: ZIP
    - » compatibility: Radix-64 encoding
  - integrated these algorithms in a easy to use application
  - distributed the code and documentation in the Internet
  - entered in agreement with a company to provide a low-cost commercial version of PGP

Security Services

- Authentication (with a signature)
- Confidentiality (the message is encrypted)
- Confidentiality and Authentication

The user chooses one of these options



Types of Keys

- (One-time) Symmetric keys
  - a different symmetric key is generated for each message
  - therefore, PGP needs a good method to generate these keys!

- Asymmetric keys
  - Several key pairs should be maintained
    - » the user might want to resort to distinct keys for different activities
    - » the user might need some old keys when accessing certain messages (e.g., while keys are being updated)
  - At least a public key has to be kept for each email peer of the user

- Symmetric keys generated from a password
  - a symmetric key created from a password is used to protect the user's private keys

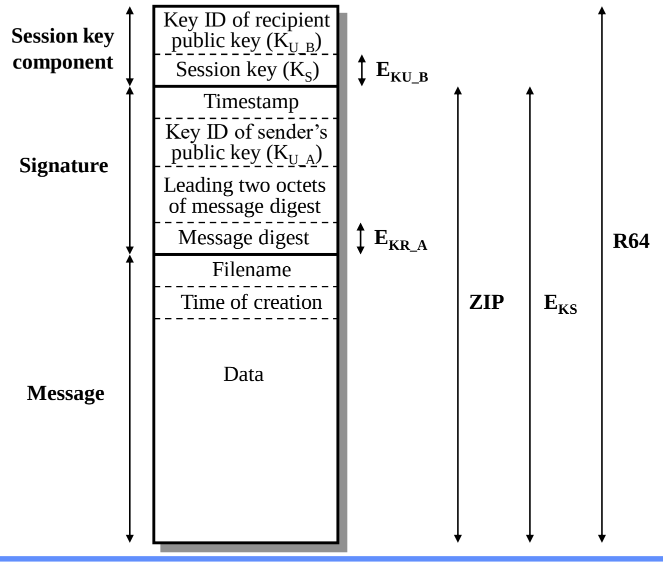
Session Key Generation

- "Real" random numbers are used to
  - **1.** create RSA key pairs
  - **2.** to provide a seed for the pseudorandom number generator
- Pseudorandom numbers are used to
  - **1.** create session keys (one for each message)
  - **2.** generate initialization vectors (IV) to be in CFB mode of encryption
- How can we generate the "real" random numbers?
  - keep a buffer with 256 random bytes
  - keep the instant, in 32-bits format, when PGP starts waiting for the keyboard to be pressed by the user
  - keep the instant when the keyboard is pressed
  - use the two instants plus the pressed letter to generate a key, which is then used to encrypt the current value of the buffer with the random bytes

Private and Public Keys Identifiers

- Since each user can have more than one private/public key pair, PGP needs to associate an identifier to each key
- One solution could be to require each user to provide the identifiers of the keys (e.g., counter + user id), but this solution creates management problems
- The solution employed by PGP consists in associating an identifier that with *high probability* is unique for each key
- The identifier of a key is equal to the less significant 64-bits of a key (i.e., key ID = K mod 2^{64})

PGP Message Format



Key Rings

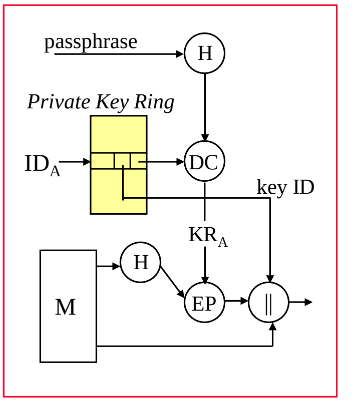
Private Key Ring

Timestamp	Key ID	Public Key	Encrypted Private Key	User ID
Ti	KUi mod 2^{64}	KUi	$E_{H\ (Pi)}$[KRi]	User i
...	...	...	...	...

Public Key Ring

Timestamp	Key ID	Public Key	Owner Trust	User ID	Key Legitimacy	Signature(s)	Signat.(s) Trust
Ti	KUi mod 2^{64}	KUi	*flag1 i*	User i	*flag2 i*		
...	...	...	...	...	...	...	...

Example : Signature



<u>*Method for storing private keys*</u>

1. the user selects a secret sentence, *passphrase*
2. use SHA-1 to generate an hash of the *passphrase*, and then delete the *passphrase*
3. encrypt the private key with CAST-128 using the hash as the key, and then remove the hash

Management of the Public Keys of Other Users

- Each user keeps a *public-key ring* with the public keys of the other users
- How can one prevent the inclusion of fake public key (e.g., C gives to A a wrong public key of B)?
  1. physically get the public key of B (e.g., with a USB pen)
  2. verify the key using the telephone (e.g., the key is sent by email, and then the hash of the key is verified by the telephone)
  3. get the B's key from a well known user D (e.g., D creates and signs a certificate with the public key of B)
  4. same as before, but one uses a CA (*Certification Authority*)

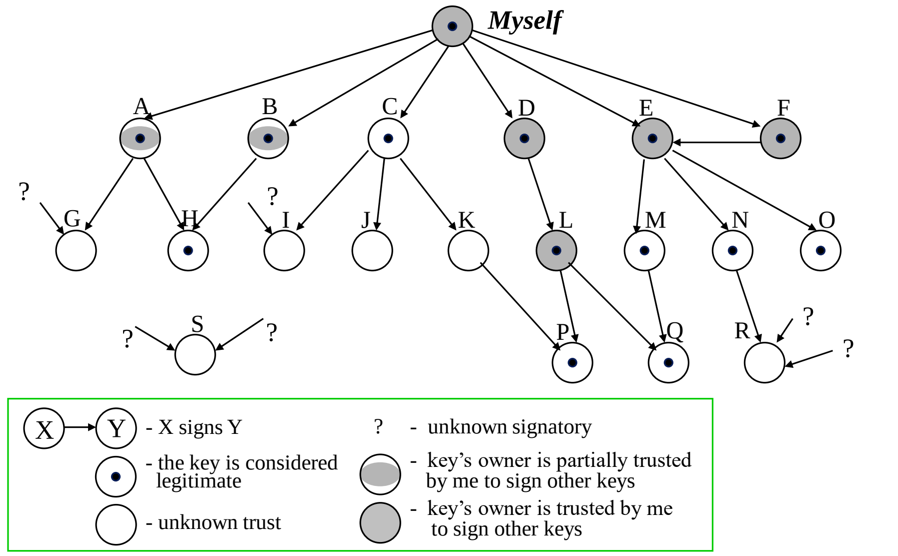
- In cases 3. and 4., the user should associate with the intermediary a level of trust => PGP employs a generic trust model based on the information provided by the other users

The Use of Trust

- Based on the following fields of the public key ring
  - **owner trust:** indicates the level of trust that the user associates with this public key **to sign other certificates** (the user provides this value)
    - **trust levels**: *unknown, untrusted, marginally trusted, completely trusted*
  - **signature trust:** for each signature on the certificate, indicates the level of trust the user associates to the signer (equal to the *owner trust* of the key used in the signature)
  - **key legitimacy:** value calculated by PGP that indicates the overall trust that this public key belongs to a certain user

- Example: When a new key is received
  - PGP asks the user to provide a level for the owner trust; if the key belongs to the user, it is assigned the *ultimate* trust
  - one or more signatures might be attached to the certificate; PGP uses the public key ring to determine the signature trust of each signature
  - the value of key legitimacy is calculated using the signature trust values; if at least one signatures has ultimate value, then legitimacy is complete; otherwise, legitimacy is equal to the weighted sum of the trust values

Example: Trust Model of PGP



Bibliography

- W. Stallings, Cryptography and Network Security (6th Edition), 2014 : chapter 19 (pag 611- 618)
- W. Stallings, Cryptography and Network Security (5th Edition), 2010 : chapter 18 (pag 592– 611)