E-MAIL SECURITY NEEDS

- **confidentiality**: protection from disclosure
- **authentication** of sender of message
- **message integrity**: protection from modification
- **non-repudiation** of origin: protection from denial by sender

SECURING E-MAIL BY PGP

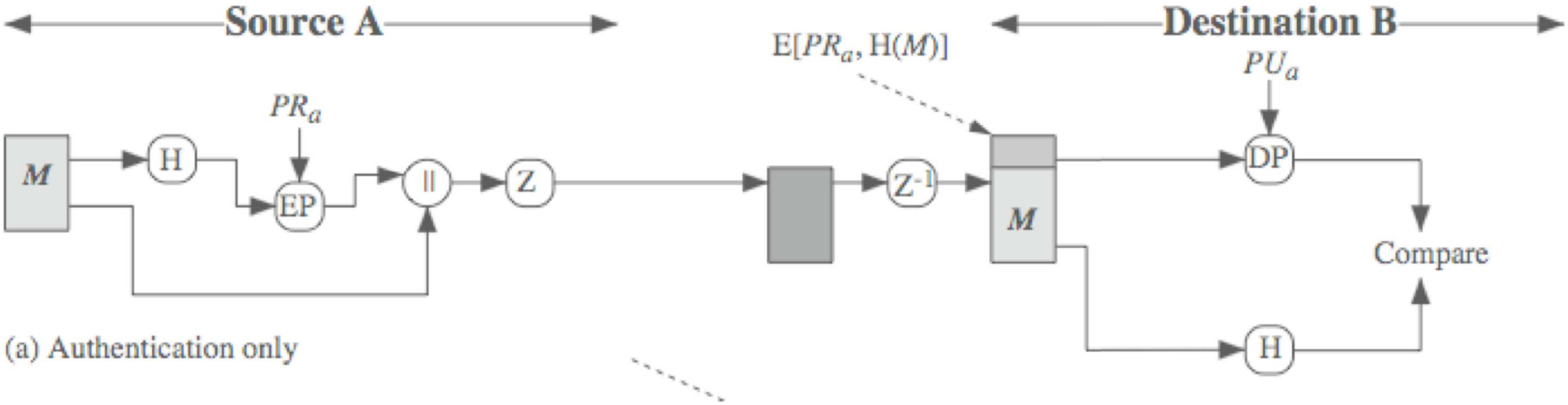
- Pretty Good Privacy is a standard created by Phil Zimmermann in 1991
  - "PGP empowers people to take their privacy into their own hands. There has been a growing social need for it. That's why I wrote it." See *Why I wrote PGP* https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html
- The slides on PGP are inspired to the well-known textbook *Cryptography and Network Security*, 5/e, by William Stallings, Chapter 18 – "Electronic Mail Security"

PRETTY GOOD PRIVACY (PGP)

- well known and widely used since the 90s
- using best available crypto algorithms
- integrated into a single program
  - Linux/Unix, PC, Macintosh and other systems
- originally free, now owned by Symantec (www.pgp.com)
- open version (OpenPGP) standardized in RCF 4880
  - several implementations, e.g., Gnu Privacy Guard (www.gnupg.org)

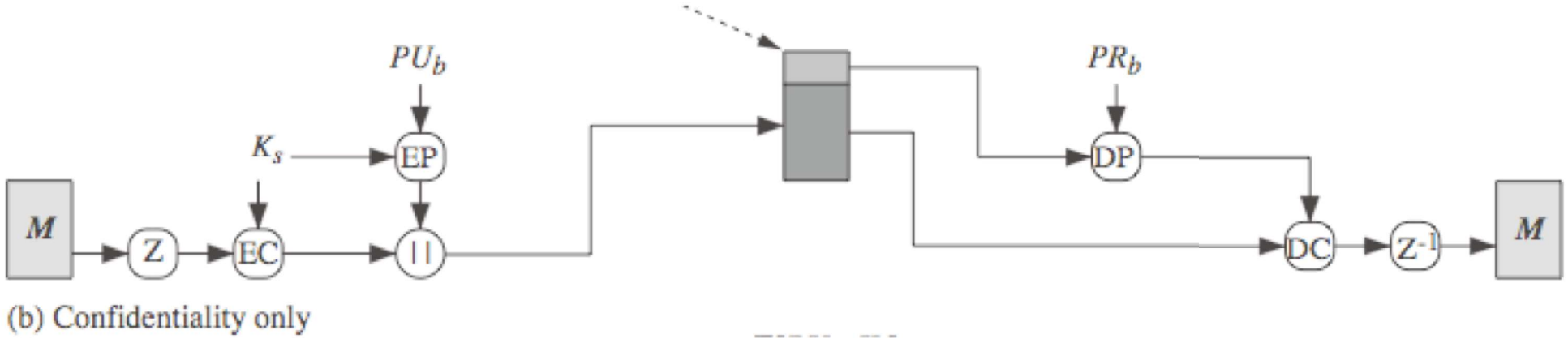
PGP AUTHENTICATION

1. sender creates message
2. make SHA-1 160-bit hash of message
3. attached RSA signed hash to message
4. receiver decrypts & recovers hash code
5. receiver verifies received message hash



PGP CONFIDENTIALITY

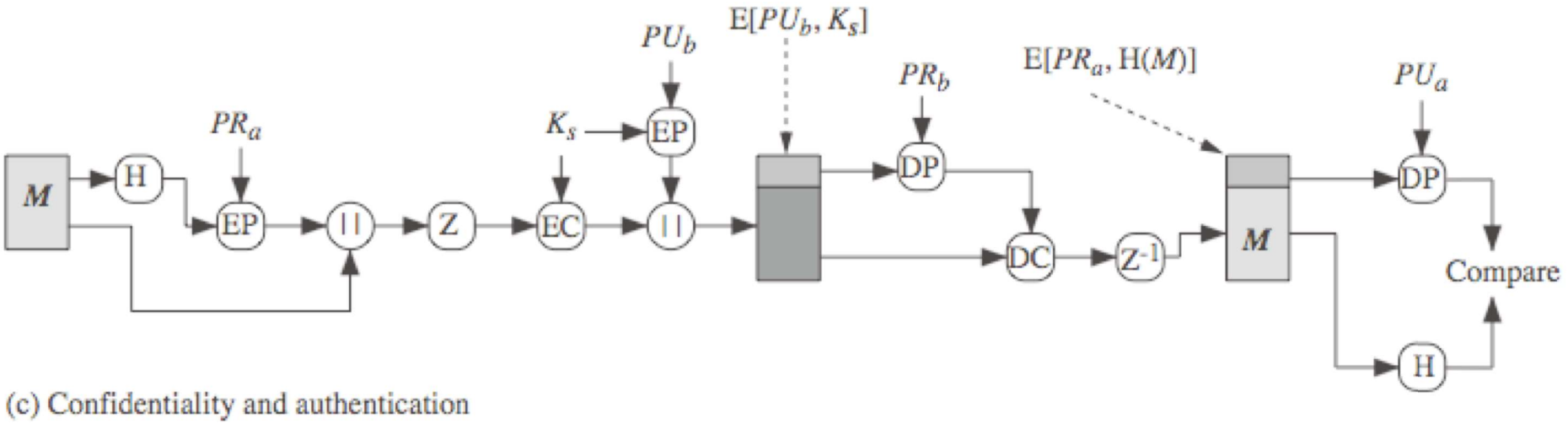
- sender forms 128-bit random session key
- encrypts message with session key
- attaches session key encrypted with RSA
- receiver decrypts & recovers session key
- session key is used to decrypt message



(b) Confidentiality only

CONFIDENTIALITY & AUTHENTICATION

- can use both services on same message
  - create signature & attach to message
  - encrypt both message & signature
  - attach RSA/ElGamal encrypted session key



(c) Confidentiality and authentication

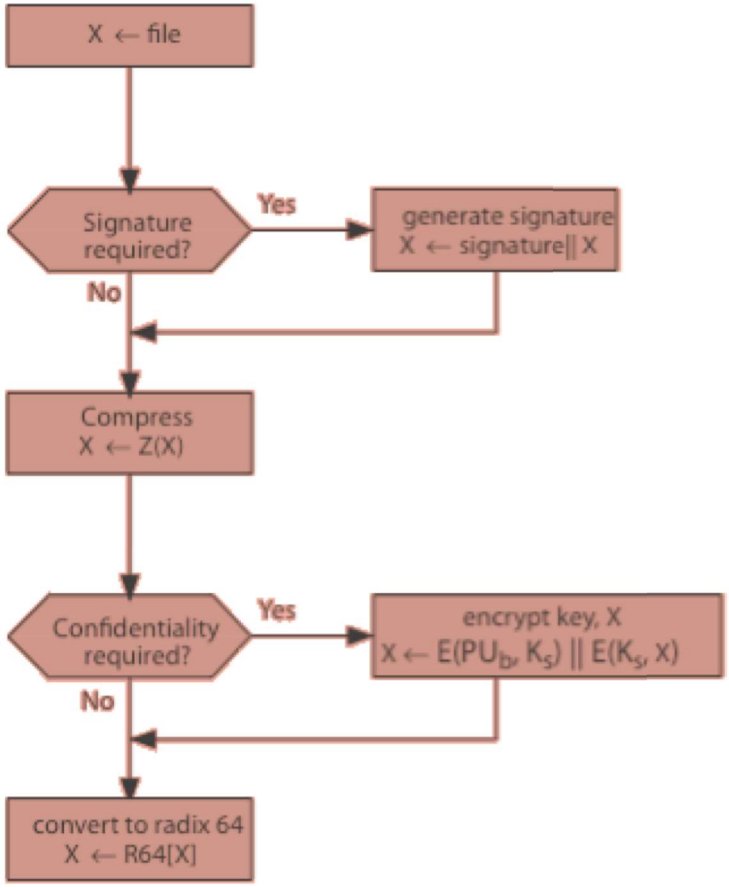
COMPRESSION

- by default PGP compresses message after signing but before encrypting
  - so can store uncompressed message & signature for later verification
  - because compression is non deterministic (if verification requires compression it may fail on legitimate messages)
- uses ZIP compression algorithm

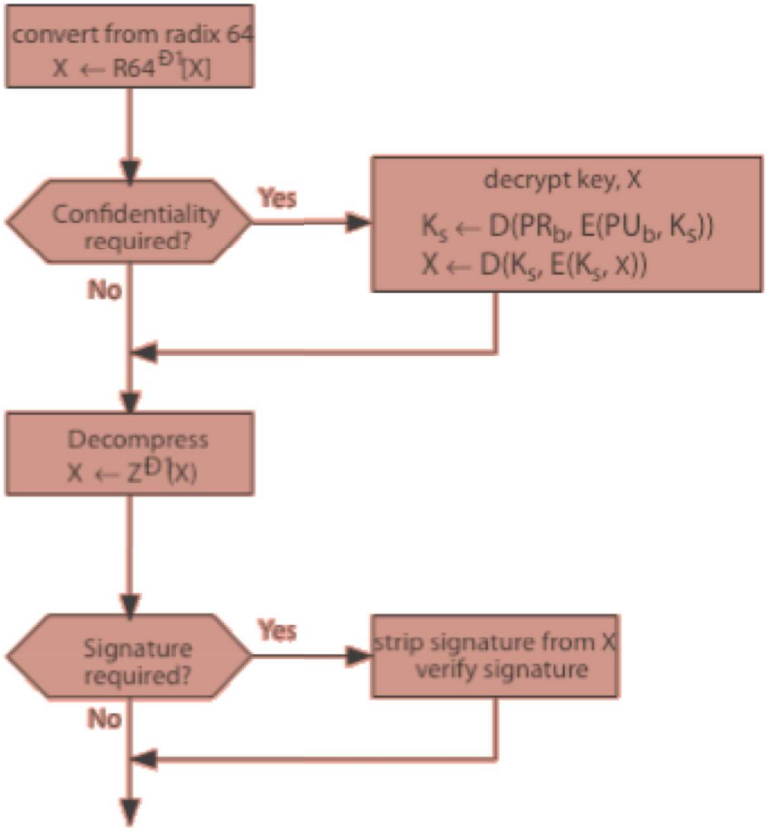
EMAIL COMPATIBILITY

- when using PGP will have binary data to send (encrypted message etc.)
- however email was designed only for text
- hence PGP must encode raw binary data into printable ASCII characters
  - uses Base-64 encoding and appends a 24-bit CRC
- PGP also segments messages if too big

PGP OPERATION – SUMMARY



(a) Generic Transmission Diagram (from A)



(b) Generic Reception Diagram (to B)

RFC 4880 - PUBLIC KEY

9.1. Public-Key Algorithms

ID	Algorithm
1	RSA (Encrypt or Sign) [HAC]
2	RSA Encrypt-Only [HAC]
3	RSA Sign-Only [HAC]
16	Elgamal (Encrypt-Only) [ELGAMAL] [HAC]
17	DSA (Digital Signature Algorithm) [FIPS186] [HAC]
18	Reserved for Elliptic Curve
19	Reserved for ECDSA
20	Reserved (formerly Elgamal Encrypt or Sign)
21	Reserved for Diffie-Hellman (X9.42, as defined for IETF-S/MIME)
100 to 110	Private/Experimental algorithm

Implementations MUST implement DSA for signatures, and Elgamal for encryption. Implementations SHOULD implement RSA keys (1). RSA Encrypt-Only (2) and RSA Sign-Only are deprecated and SHOULD NOT be generated, but may be interpreted. See Section 13.5. See Section 13.8 for notes on Elliptic Curve (18), ECDSA (19), Elgamal Encrypt or Sign (20), and X9.42 (21). Implementations MAY implement any other algorithm.

RFC 4880 - SYMMETRIC KEY

9.2. Symmetric-Key Algorithms

ID	Algorithm
0	- Plaintext or unencrypted data
1	- IDEA [IDEA]
2	- TripleDES (DES-EDE, [SCHNEIER] [HAC] - 168 bit key derived from 192)
3	- CAST5 (128 bit key, as per [RFC2144])
4	- Blowfish (128 bit key, 16 rounds) [BLOWFISH]
5	- Reserved
6	- Reserved
7	- AES with 128-bit key [AES]
8	- AES with 192-bit key
9	- AES with 256-bit key
10	- Twofish with 256-bit key [TWOFISH]
100 to 110	- Private/Experimental algorithm

Implementations MUST implement TripleDES. Implementations SHOULD implement AES-128 and CAST5. Implementations that interoperate with PGP 2.6 or earlier need to support IDEA, as that is the only symmetric cipher those versions use. Implementations MAY implement any other algorithm.

added support for Camelia in RFC 5581 (2009)

RFC 4880 - HASH FUNCTIONS

9.4. Hash Algorithms

ID	Algorithm	Text Name
1	- MD5 [HAC]	"MD5"
2	- SHA-1 [FIPS180]	"SHA1"
3	- RIPE-MD/160 [HAC]	"RIPEMD160"
4	- Reserved	
5	- Reserved	
6	- Reserved	
7	- Reserved	
8	- SHA256 [FIPS180]	"SHA256"
9	- SHA384 [FIPS180]	"SHA384"
10	- SHA512 [FIPS180]	"SHA512"
11	- SHA224 [FIPS180]	"SHA224"
100 to 110	- Private/Experimental algorithm	

Implementations MUST implement SHA-1. Implementations MAY implement other algorithms. MD5 is deprecated.

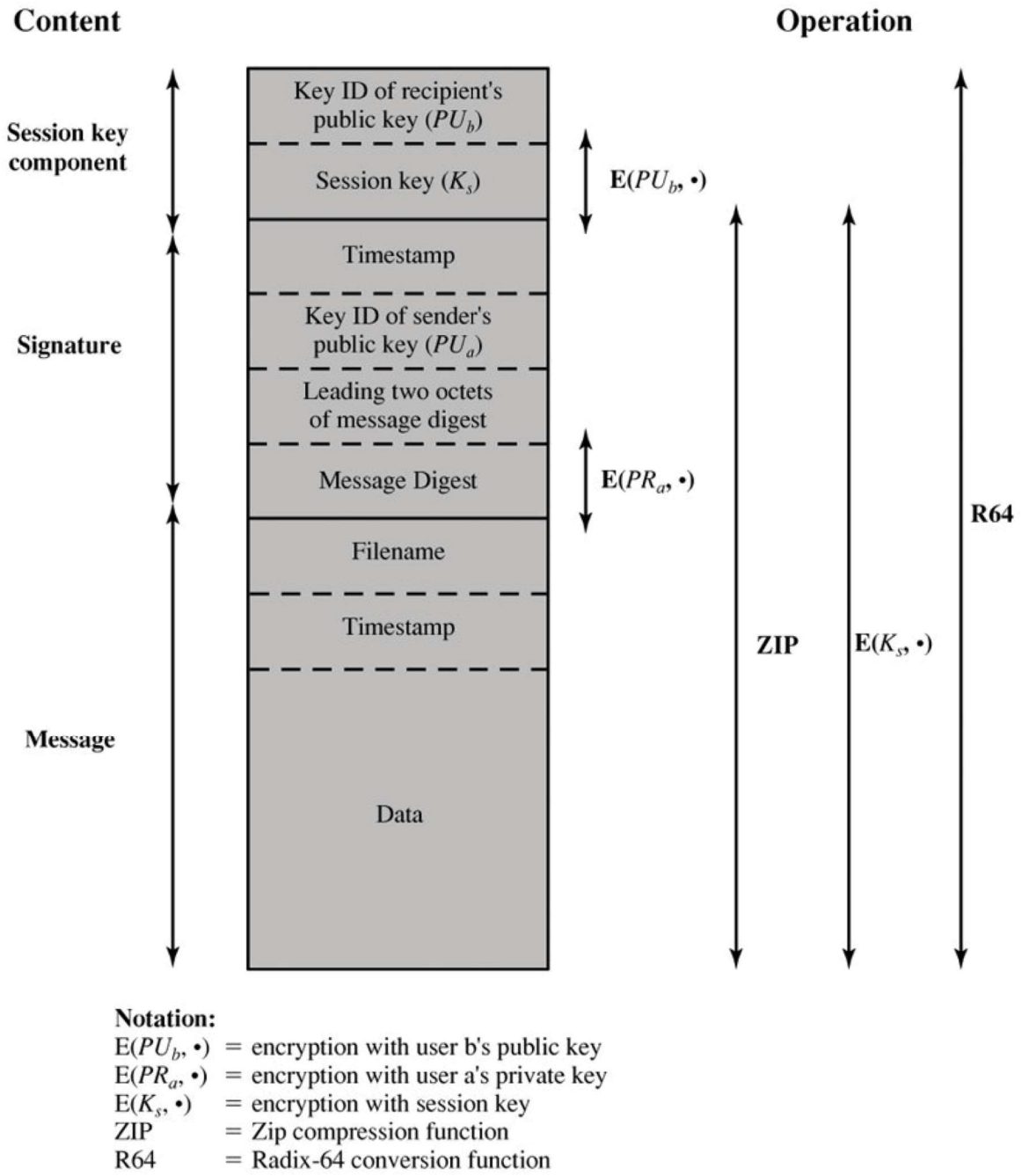
PGP SESSION KEYS

- PGP needs a session key for each message
- PGP maintains a 256-byte buffer of random bits. Each time PGP expects a keystroke, it records the time, in 32-bit format, at which it starts waiting. When it receives the keystroke, it records the time the key was pressed and the 8-bit value of the keystroke. The time and keystroke information are used to generate a key, which is, in turn, used to encrypt the current value of the random-bit buffer
- Pseudorandom number generation makes use of a 24-octet seed and produces a 16-octet session key, an 8-octet initialization vector, and a new seed to be used for the next pseudorandom number generation.
- The algorithm is based on the X9.17 algorithm but uses CAST-128 instead of triple DES for encryption

PGP PUBLIC & PRIVATE KEYS

- since many public/private keys may be in use (by one user), need to identify which is actually used to encrypt session key in a message
  - could send full public-key with every message
  - but this is inefficient
- rather use a key identifier (ID) based on key
  - is least significant 64-bits of the key
  - will very likely be unique
- also use key ID in signatures

PGP MESSAGE FORMAT



Notation:

E(PU_b, •)	=	encryption with user b's public key
E(PR_a, •)	=	encryption with user a's private key
E(K_s, •)	=	encryption with session key
ZIP	=	Zip compression function
R64	=	Radix-64 conversion function

PGP KEY RINGS

- each PGP user has a pair of keyrings:
  - public-key ring contains all the public-keys of other PGP users known to this user, indexed by key ID
  - private-key ring contains the public/private key pair(s) for this user, indexed by key ID & encrypted keyed from a hashed passphrase
- security of private keys thus depends on the passphrase security

PGP KEY RINGS

Private Key Ring

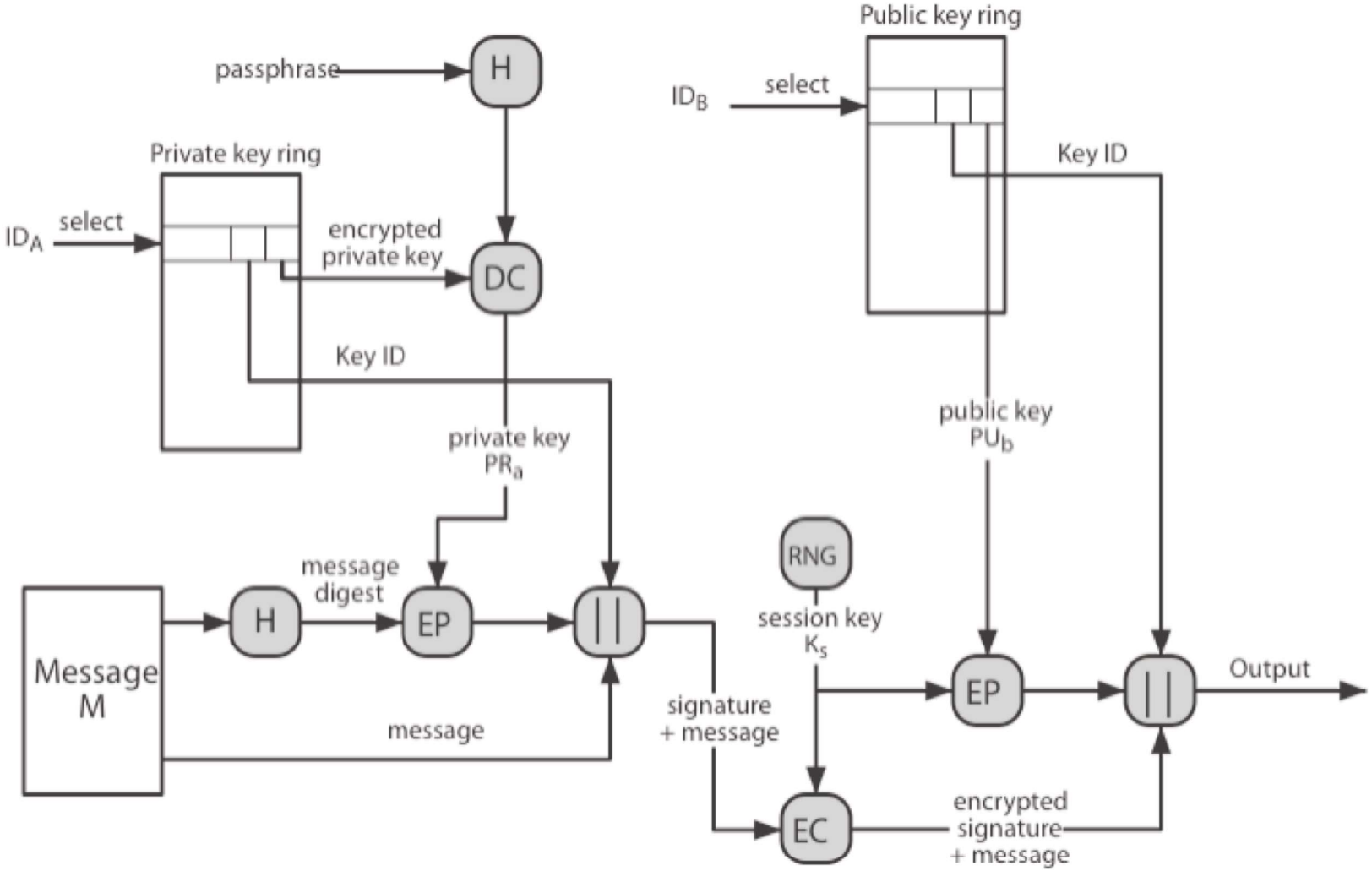
Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•
Ti	PU_i mod 2^{64}	PU_i	E(H(P_i), PR_i)	User *i*
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•

Public Key Ring

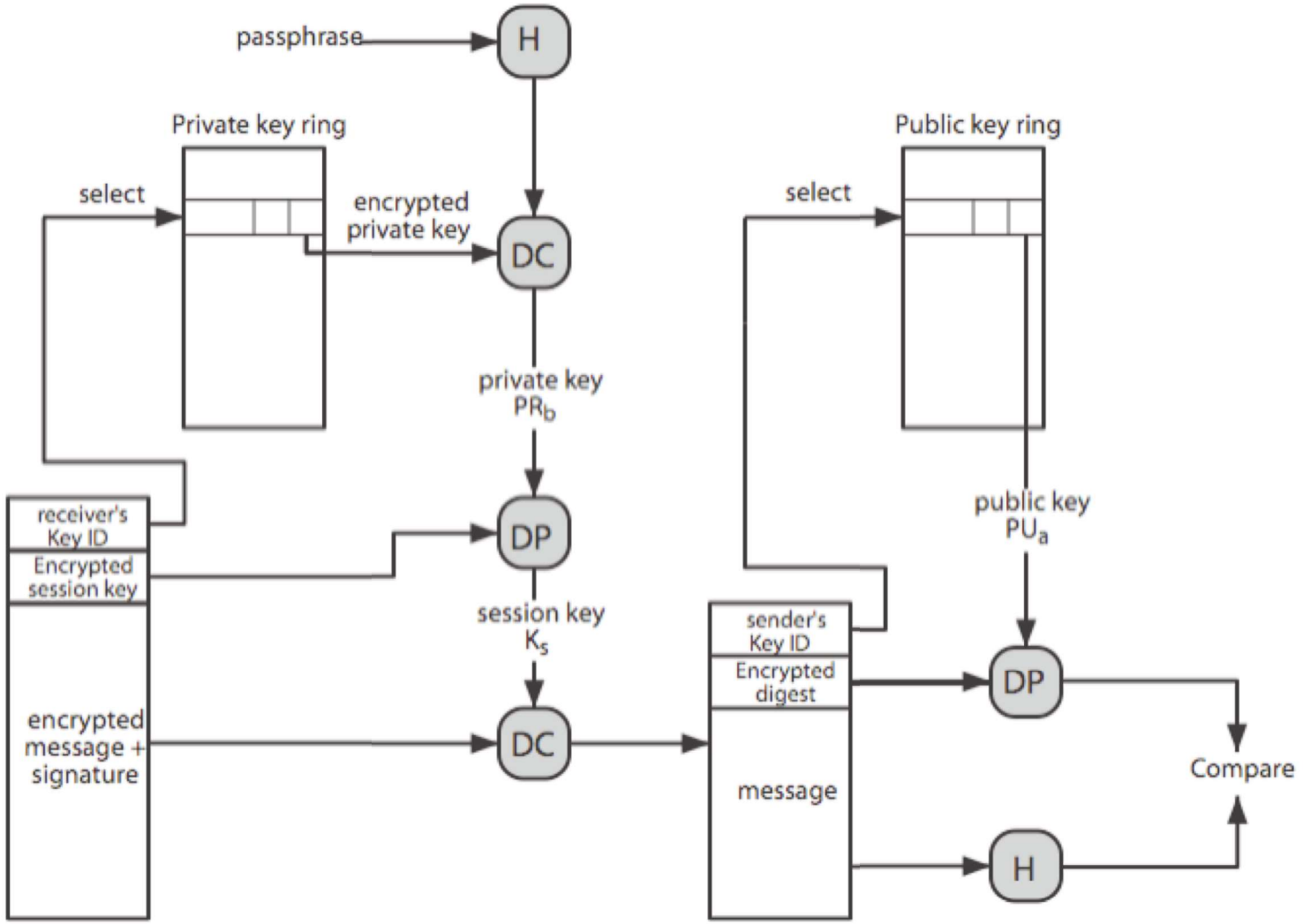
Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
T_i	PU_i mod 2^{64}	PU_i	trust_flag$_i$	User *i*	trust_flag$_i$		
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•

* = field used to index table

PGP MESSAGE GENERATION



PGP MESSAGE RECEPTION



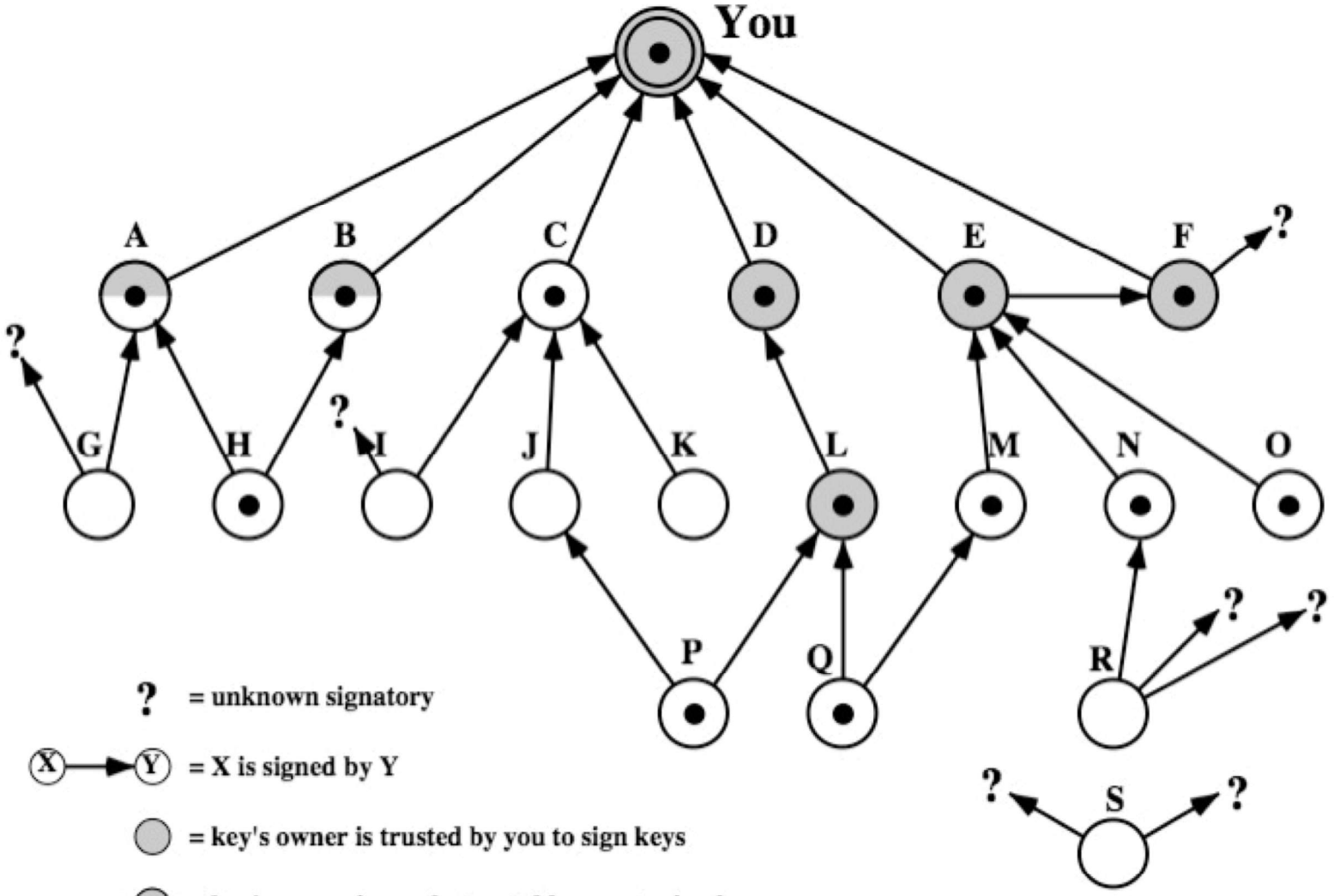
PGP KEY MANAGEMENT

- rather than relying on certificate authorities
- in PGP every user is own CA
  - can sign keys for users they know directly
- forms a "web of trust"
  - trust keys have signed
  - can trust keys others have signed if have a chain of signatures to them
- key ring includes trust indicators
- users can also revoke their keys
- a possible key-sign procedure http://herrons.com/keysigning-party-guide/

WEB OF TRUST (ZIMMERMANN)

As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys.

PGP TRUST MODEL EXAMPLE



PGP TODAY

- OpenPGP is an Internet standard (RFC 4880, 2007)
- many e-mail clients provide OpenPGP-compliant email security
- best known implementations of OpenPGP
  - PGP by Symantec Inc.
  - GNU Privacy Guard (GnuPG or GPG) by The Free Software Foundation. Open-source

ENIGMAIL EXAMPLE

- supported email clients
  - Mozilla Thunderbird
  - Mozilla SeaMonkey
  - Eudora 1.0 OSE
  - Postbox
- GnuPG Software
- EnigMail plugin (http://enigmail.mozdev.org)
  - language packs available


Gestione componenti aggiuntivi
Posta in arrivo - DIS
Gestione componenti aggiuntivi
Cerca tra i componenti aggiuntivi
Esplora
Estensioni
Aspetto
Plugin
Enigmail 1.4.1
di Patrick Brunschwig
OpenPGP message encryption and authentication
Enigmail adds OpenPGP message encryption and authentication to your email client. It features automatic encryption, decryption and integrated key management functionality.
Enigmail requires GnuPG (www.gnupg.org) for the cryptographic functions. Note: GnuPG is not part of the installation.
The addon offered here supports Windows, Linux (32 and 64-bit) and Mac OS X. Versions for more platforms are available from the homepage.
Aggiornamento automatico
Predefinito
Attivo
Disattivato
Ultimo aggiornamento
21 aprile 2012
Sito web
http://enigmail.mozdev.org/
Voto
133 recensioni
Preferenze
Disattiva
Rimuovi