"I should be able to whisper something in your ear, even if your ear is 1000 miles away, and the government disagrees with that."

-- Philip Zimmermann

PGP - Pretty Good Privacy

- services
- message format
- random number generation
- key and trust mamagement

What is PGP?

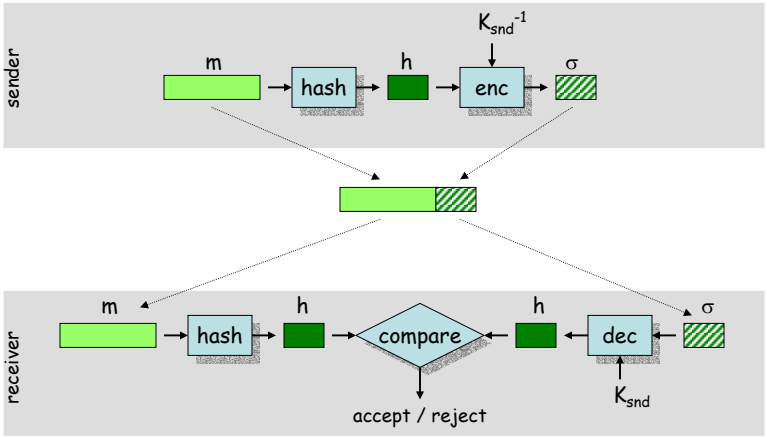
- general purpose application to protect (encrypt and/or sign) files
- can be used to protect e-mail messages
- can be used by corporations as well as individuals
- based on strong cryptographic algorithms (IDEA, RSA, SHA-1)
- first version developed by Phil Zimmermann
- international version available free of charge at http://www.pgpi.org

PGP services

- messages
  - authentication
  - confidentiality
  - compression
  - e-mail compatibility
  - segmentation and reassembly
- key management
  - generation, distribution, and revocation of public/private keys
  - generation and transport of session keys and IVs

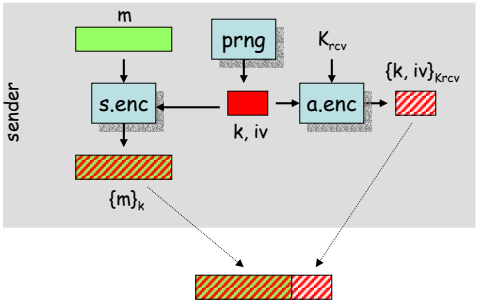
Message authentication

- based on digital signatures
- supported algorithms: RSA/SHA and DSS/SHA



Message confidentiality

- symmetric key encryption in CFB mode with a random session key and IV
- session key and IV is encrypted with the public key of the receiver
- supported algorithms:
  - symmetric: CAST, IDEA, 3DES
  - asymmetric: RSA, ElGamal



PGP services

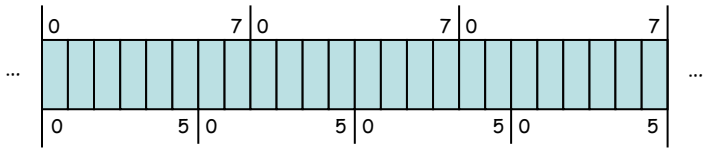
Compression

- applied after the signature
  - enough to store clear message and signature for later verification
  - it would be possible to dynamically compress messages before signature verification, but …
    - then all PGP implementations should use the same compression algorithm
    - however, different PGP versions use slightly different compression algorithms
- applied before encryption
  - would be useless after encryption
  - compression reduces redundancy → makes cryptanalysis harder
- supported algorithm: ZIP

PGP services

E-mail compatibility

- encrypted messages and signatures may contain arbitrary octets
- most e-mail systems support only ASCII characters
- PGP converts an arbitrary binary stream into a stream of printable ASCII characters
- radix 64 conversion: 3 8-bit blocks → 4 6-bit blocks



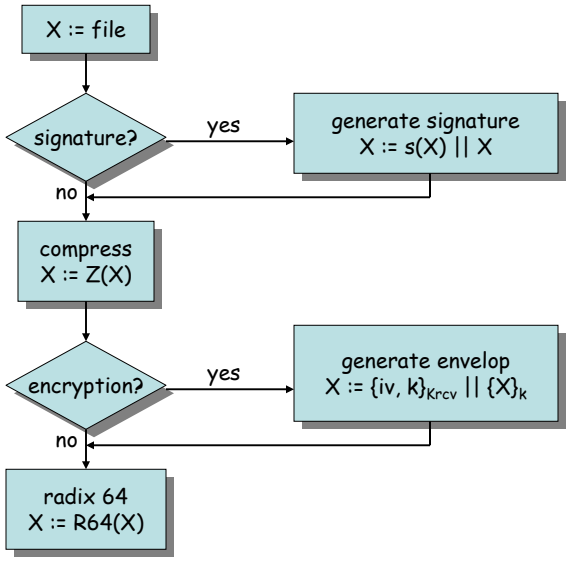
PGP services

Radix 64 conversion table

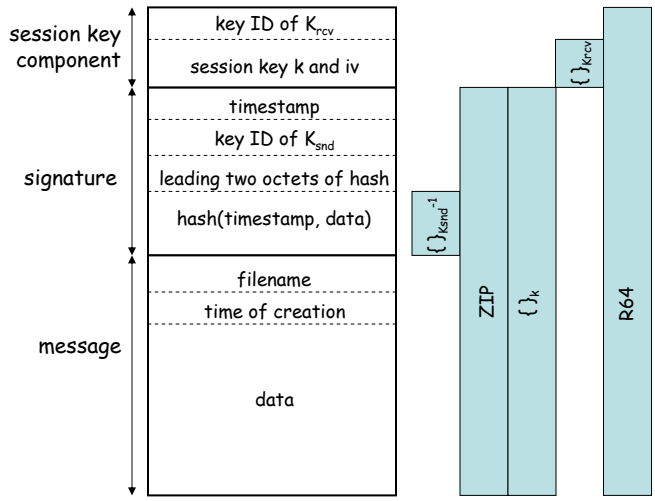
6-bit value	character encoding	6-bit value	character encoding	6-bit value	character encoding	6-bit value	character encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
						(pad)	=

PGP services

Combining services



General PGP message format



PGP message format

Key IDs

- a user may have several public key – private key pairs
  - which private key to use to decrypt the session key?
  - which public key to use to verify a signature?
- transmitting the whole public key would be wasteful
- associating a random ID to a public key would result in management burden
- PGP key ID: least significant 64 bits of the public key
  - unique within a user with very high probability

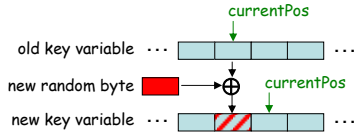
PGP random number generation

Random numbers in PGP

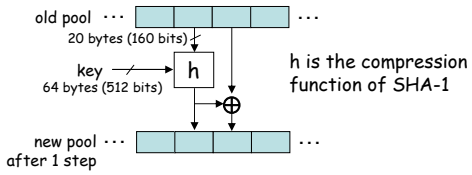
- pseudo-random numbers generated by a PRNG
  - used as session keys and IVs
- true random numbers
  - provide the initial seed for the PRNG
  - provide additional input during pseudo-random number generation
  - used to generate public key – private key pairs

The true random pool

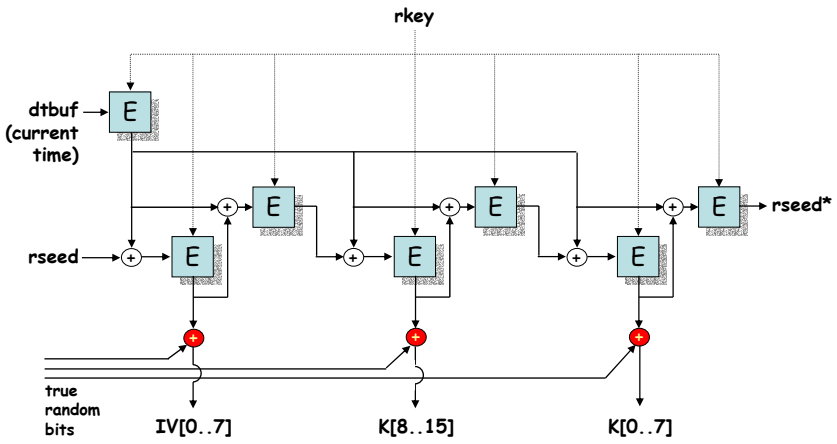
- PGP maintains a 640-byte *pool* of random bits and a 64 byte *key* variable
- new random samples are obtained from keystroke timings and mouse movement and mixed into the pool as follows:
  - each new random byte is XORed into the key variable



  - once enough new bytes have been added to the key, it is used to "encrypt" the pool in a "message digest cipher" configuration in CBC mode



Generation of session keys and IVs



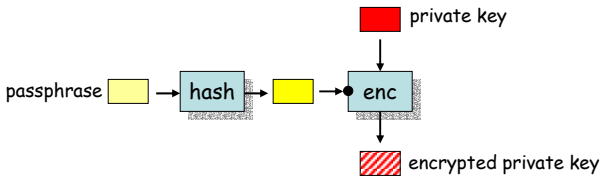
- the PRNG is based on X9.17 extended with a pre-wash and a post-wash operation (see next slide)
- instead of 3DES, PGP uses CAST-128

Generation of session keys and IVs

- pre-wash
  - take the hash of the message
    - this has already been generated if the message is being signed
    - otherwise the first 4K of the message is hashed
  - use the result as a key, use a null IV, and encrypt [rkey, rseed] in CFB mode
    - if [rkey, rseed] is empty, it is filled up with true random bits from the random pool
  - set [rkey, rseed] to the result of the encryption
- post-wash
  - generate 24 more bytes as before but without XORing in true random bytes
  - encrypt the result in CFB mode using K and IV
  - set [rkey, rseed] to the result of the encryption

Private-key ring

- used to store the public key – private key pairs owned by a given user
- essentially a table, where each row contains the following entries:
  - timestamp
  - key ID (indexed)
  - public key
  - encrypted private key
  - user ID (indexed)



Public-key ring

- used to store public keys of other users
- a table, where each row contains the following entries:
  - timestamp
  - key ID (indexed)
  - public key
  - user ID (indexed)
  - owner trust
  - signature(s)
  - signature trust(s)
  - key legitimacy

Trust management

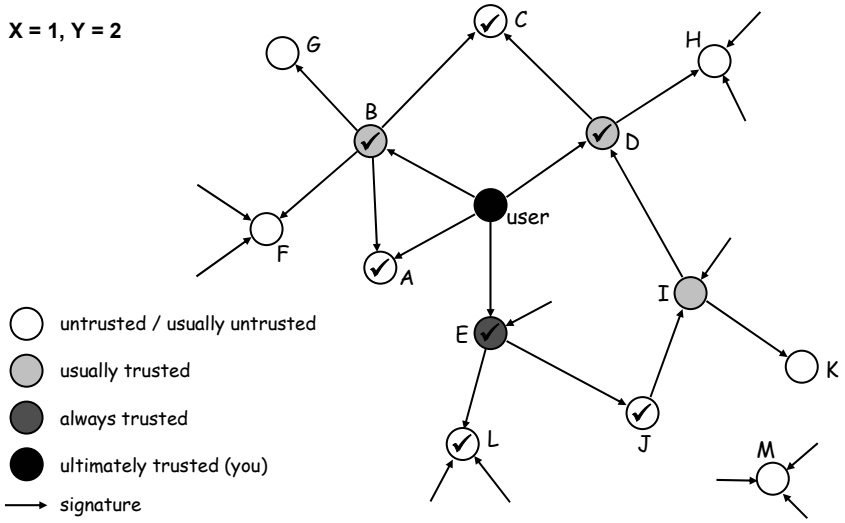
- owner trust
  - assigned by the user
  - possible values:
    - *unknown user*
    - *usually not trusted to sign*
    - *usually trusted to sign*
    - *always trusted to sign*
    - *ultimately trusted* (own key, present in private key ring)
- signature trust
  - assigned by the PGP system
  - if the corresponding public key is already in the public-key ring, then its owner trust entry is copied into signature trust
  - otherwise, signature trust is set to *unknown user*

Trust management

- key legitimacy
  - computed by the PGP system
  - if at least one signature trust is ultimate, then the key legitimacy is 1 (complete)
  - otherwise, a weighted sum of the signature trust values is computed
    - always trusted signatures has a weight of 1/X
    - usually trusted signatures has a weight of 1/Y
    - X, Y are user-configurable parameters
  - example: X=2, Y=4
    - 1 ultimately trusted, or
    - 2 always trusted, or
    - 1 always trusted and 2 usually trusted, or
    - 4 usually trusted signatures are needed to obtain full legitimacy

Example – key legitimacy

X = 1, Y = 2



Public-key revocation

- why to revoke a public key?
  - suspected to be compromised (private key got known by someone)
  - re-keying
- the owner issues a revocation certificate …
  - has a similar format to normal public-key certificates
  - contains the public key to be revoked
  - signed with the corresponding private key
- and disseminates it as widely and quickly as possible
- if a key is compromised:
  - e.g., Bob knows the private key of Alice
  - Bob can issue a revocation certificate to revoke the public key of Alice
  - even better for Alice

Recommended readings

- Philip Zimmermann on PGP (part of the documentation that comes with the software)
- docs and faqs on http://www.pgpi.org/