Modernizing the OpenPGP Message Format

draft-ford-openpgp-format-00

Bryan Ford
Swiss Federal Institute of Technology (EPFL)

IETF 94 – November 3, 2015

Possible Goals for Discussion

- Modernize cryptographic suite
  - Deprecate SHA-1, shift to authenticated encryption
- Metadata protection for encrypted files
  - Leave no byte unencrypted
  - Padding to minimize leakage via length
- Partial-file integrity protection [DKG, CFRG list]
  - Streaming-mode incremental integrity checking
  - Integrity-protected random access
- Others???

Cryptographic Suite

Modernizing cipher suite, especially MACs

- Ditch SHA-1, Modification Detection packet
- Support authenticated encryption (AEAD)

Which scheme(s)? Some options:

- AES-GCM: well-established, safe if not shiny
- Keccak/SHA-3 sponge: newly standardized
- ChaCha20-Poly1305: fast, popular “alt-crypt”
- Future: CAESAR competition winner, finalist(s)

What about passphrase? Adopt scrypt/???

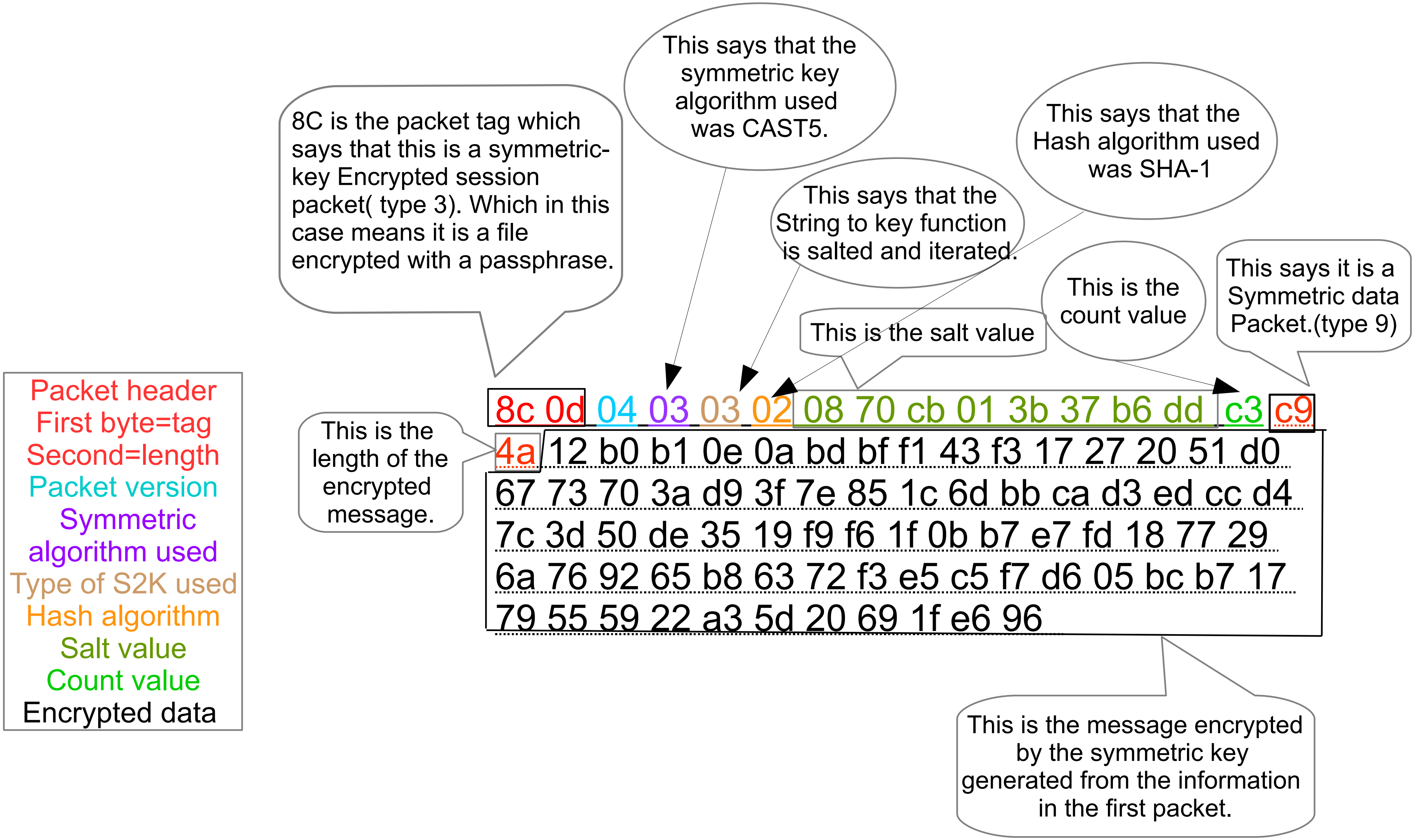
Cryptographic Suite

Some technical format issues:

- Repurpose existing packets (tags 18+19)
  or define new AEAD-protected packet (tag 20?)
- Merge MAC check into encrypted data packet?
  Safe to assume MACs are always fixed-length?
- AEAD nonce: always explicitly transmitted?
  Implicitly defined (e.g., by counting within file)?
- “Additional Data” (AD): any use in OpenPGP?

Metadata Protection

Should encrypted files leak all this metadata?



8C is the packet tag which says that this is a symmetric-key Encrypted session packet(type 3). Which in this case means it is a file encrypted with a passphrase.

This says that the symmetric key algorithm used was CAST5.

This says that the String to key function is salted and iterated.

This says that the Hash algorithm used was SHA-1

This is the salt value

This is the count value

This says it is a Symmetric data Packet.(type 9)

This is the length of the encrypted message.

```
8c 0d 04 03 03 02 08 70 cb 01 3b 37 b6 dd c3 c9
4a 12 b0 b1 0e 0a bd bf f1 43 f3 17 27 20 51 d0
67 73 70 3a d9 3f 7e 85 1c 6d bb ca d3 ed cc d4
7c 3d 50 de 35 19 f9 f6 1f 0b b7 e7 fd 18 77 29
6a 76 92 65 b8 63 72 f3 e5 c5 f7 d6 05 bc b7 17
79 55 59 22 a3 5d 20 69 1f e6 96
```

This is the message encrypted by the symmetric key generated from the information in the first packet.

Packet header
First byte=tag
Second=length
Packet version
Symmetric algorithm used
Type of S2K used
Hash algorithm
Salt value
Count value
Encrypted data

Metadata Protection

Metadata that might be useful to (some) attackers:

- Magic: this is an OpenPGP file! Suspicious!
- Cipher: is it worth trying to crack?
- Passphrase: worth trying password cracker?
- Recipient key-IDs: where to point rubber hose?
- # of recipients: aha, it's *that* group of dissidents!

Metadata Protection

Set goal to “encrypted every bit”?

- Produce Uniform Random Blobs (URBs)

Technical+usability challenges:

- How does recipient find, decrypt session key?
  - Obviously *requires* “trial decryptions”; fast enough?
- How to efficiently handle *multiple*
  - Passphrases
  - Receipient public-keys
  - Public-key schemes, curves

Claim: all are manageable. But how worthwhile?

What about Padding?

Encrypted file length leaks metadata too!

- Straw-man: pad all encrypted files to same size
  - Reduces information leakage to zero, yay!
- Wood-man: pad to next power of two
  - Reduces leakage from O(log L) to O(log log L)
  - “Best possible” while tolerating constant-factor waste
- Brick-man: pad a bit more intelligently
  - Still reduce leakage from O(log L) to O(log log L)
  - But limit waste to 12% max, decreasing with file size
  - Details in draft-in-progress, will share on request

Encrypted file size vs padding waste

Length	Length bits	Leak bits	Length inc	Max waste
1	1	0	1	0.00%
2	2	1	1	0.00%
4	3	2	1	0.00%
8	4	2	2	11.11%
16	5	3	2	5.88%
32	6	3	4	9.09%
64	7	3	8	10.77%
128	8	3	16	11.63%
256	9	4	16	5.84%
512	10	4	32	6.04%
1024	11	4	64	6.15%
2048	12	4	128	6.20%
4096	13	4	256	6.22%
8192	14	4	512	6.24%
16384	15	4	1024	6.24%
32768	16	4	2048	6.25%
65536	17	5	2048	3.12%
131072	18	5	4096	3.12%
262144	19	5	8192	3.12%
524288	20	5	16384	3.12%
1048576	21	5	32768	3.12%
2097152	22	5	65536	3.12%
4194304	23	5	131072	3.12%
8388608	24	5	262144	3.12%

Partial-File Integrity Protection

Brought up by DKG, discussed on CFRG list.

Two motivating use-cases: (orthogonal?)

- Streaming-mode decryption (restore backup)
  - Check bytes before they leave pipe, w/o storing it all
  - Need incremental MAC+signature per chunk?
- Random-access decryption (ala Tahoe-LAFS)
  - Encryptor builds Merkle tree, stores in trailer
  - Decryptor uses to decrypt, check individual chunks
- If we support, are they mandatory? Optional?
  - Simplicity vs power vs usability?