Appendix P
PGP Random Number Generation

William Stallings
Copyright 2010

P.1 TRUE RANDOM NUMBERS 2
P.2 PSEUDORANDOM NUMBERS 2

Supplement to
Cryptography and Network Security, Fifth Edition
William Stallings
Prentice Hall 2010
ISBN-10: 0136097049
http://williamstallings.com/Crypto/Crypto5e.html

PGP uses a complex and powerful scheme for generating random numbers and pseudorandom numbers, for a variety of purposes. PGP generates random numbers from the content and timing of user keystrokes, and pseudorandom numbers using an algorithm based on the one in ANSI X9.17. PGP uses these numbers for the following purposes:

- True random numbers:
  - —used to generate RSA key pairs
  - —provide the initial seed for the pseudorandom number generator
  - —provide additional input during pseudorandom number generation
- Pseudorandom numbers:
  - —used to generate session keys
  - —used to generate initialization vectors (IVs) for use with the session key in CFB mode encryption

P.1 TRUE RANDOM NUMBERS

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.

P.2 PSEUDORANDOM NUMBERS

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 described in Chapter 7 (see Figure 7.4) but uses CAST-128 instead of triple DES for encryption. The algorithm uses the following data structures:

1. Input

—randseed.bin (24 octets): If this file is empty, it is filled with 24 true random octets.

—message: The session key and IV that will be used to encrypt a message are themselves a function of that message. This further contributes to the randomness of the key and IV, and if an opponent already knows the plaintext content of the message, there is no apparent need for capturing the one-time session key.

2. Output

—K (24 octets): The first 16 octets, K[0..15], contain a session key, and the last eight octets, K[16..23], contain an IV.

—randseed.bin (24 octets): A new seed value is placed in this file.

3. Internal data structures

—dtbuf (8 octets): The first 4 octets, dtbuf[0..3], are initialized with the current date/time value. This buffer is equivalent to the DT variable in the X12.17 algorithm.

—rkey (16 octets): CAST-128 encryption key used at all stages of the algorithm.

—rseed (8 octets): Equivalent to the X12.17 V_i variable.

—rbuf (8 octets): A pseudorandom number generated by the algorithm. This buffer is equivalent to the X12.17 R_i variable.

—K' (24 octets): Temporary buffer for the new value of randseed.bin.

The algorithm consists of nine steps, G1 through G9. The first and last steps are obfuscation steps, intended to reduce the value of a captured randseed.bin file to an opponent. The remaining steps are essentially equivalent to three iterations of the X12.17 algorithm and are illustrated in Figure P.1 (compare Figure 7.4). To summarize,

G1. [Prewash previous seed]

a. Copy randseed.bin to K[0..23].

b. Take the hash of the message (this has already been generated if the message is being signed; otherwise the first 4K octets of the message are used). Use the result as a key, use a null IV, and encrypt K in CFB mode; store result back in K.

G2. [Set initial seed]

a. Set dtbuf[0..3] to the 32-bit local time. Set dtbuf[4..7] to all zeros. Copy rkey ← K[0..15]. Copy rseed ← K[16..23].

b. Encrypt the 64-bit dtbuf using the 128-bit rkey in ECB mode; store the result back in dtbuf.

G3. **[Prepare to generate random octets]** Set rcount ← 0 and k ← 23. The loop of steps G4-G7 will be executed 24 times (k = 23...0), once for each random octet produced and placed in K. The variable rcount is the number of unused random octets in rbuf. It will count down from 8 to 0 three times to generate the 24 octets.

G4. **[Bytes available?]** If rcount = 0 goto G5 else goto G7. Steps G5 and G6 perform one instance of the X12.17 algorithm to generate a new batch of eight random octets.

G5. **[Generate new random octets]**

a. rseed ← rseed ⊕ dtbuf

b. rbuf ← E_{rkey}[rseed] in ECB mode

G6. **[Generate next seed]**

a. rseed ← rbuf ⊕ dtbuf

b. rseed ← E_{rkey}[rseed] in ECB mode

c. Set rcount ← 8

G7. **[Transfer one byte at a time from rbuf to K]**

a. Set rcount ← rcount – 1

b. Generate a true random byte b, and set K[k] ← rbuf[rcount] ⊕ b

G8. **[Done?] If k = 0 goto G9 else set k ← k – 1 and goto G4**

G9. **[Postwash seed and return result]**

a. Generate 24 more bytes by the method of steps G4-G7, except do not XOR in a random byte in G7. Place the result in buffer K'

b. Encrypt K' with key K[0..15] and IV K[16..23] in CFB mode; store result in randseed.bin

c. Return K

It should not be possible to determine the session key from the 24 new octets generated in step G9.a. However, to make sure that the stored randseed.bin file provides no information about the most recent session key, the 24 new octets are encrypted and the result is stored as the new seed.

This elaborate algorithm should provide cryptographically strong pseudorandom numbers.

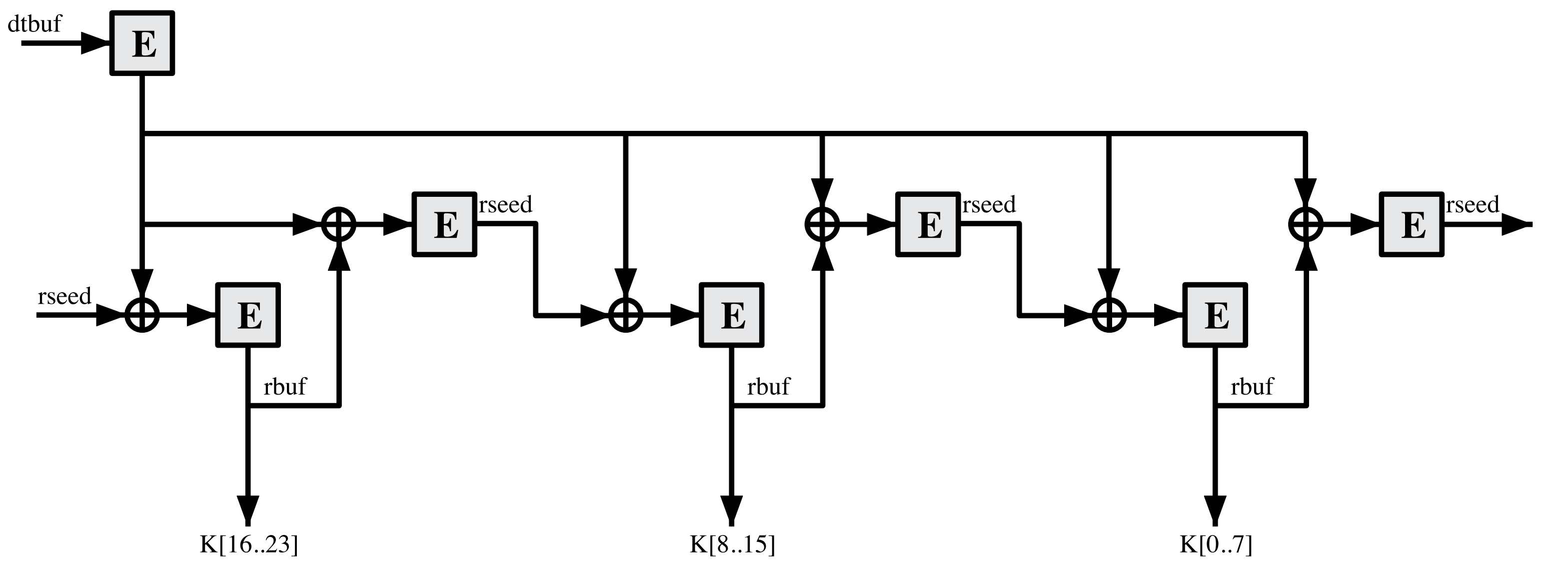

Figure P.1 PGP Session Key and IV Generation (steps G2 through G8)