Computer and Network Security: Random Number Generators

Kameswari Chebrolu

Usage

- Secure communication requires high quality (pseudo) random numbers
  - One time pads; stream ciphers (symmetric key)
  - RSA public key (generate and test for primality)
  - Salts in hashing
  - Session keys; nonces during handshaking
- Note: Security requirements not the same across all use cases

Truly Random Requirements

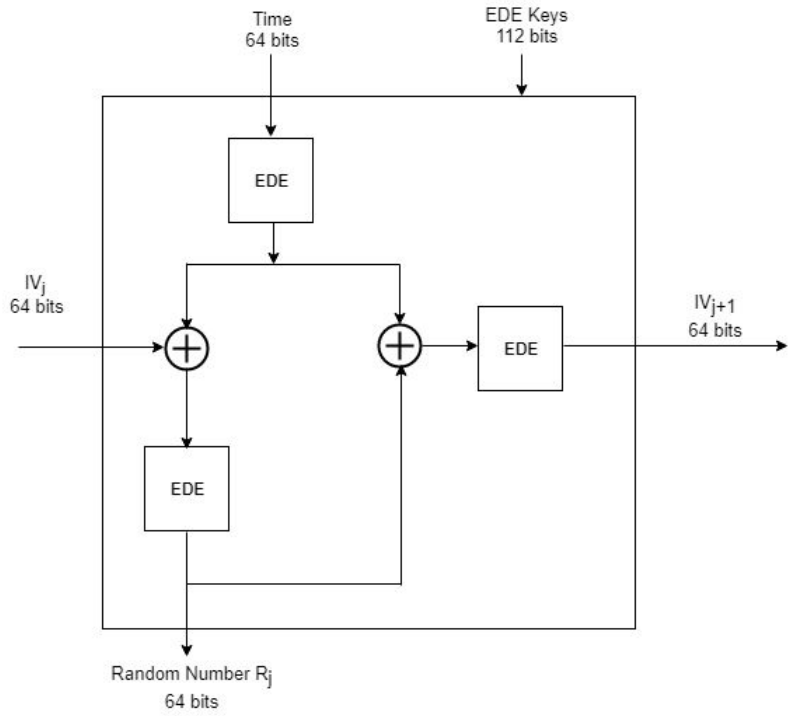
- Uniform Distribution: all numbers in given range occur equally often
- Independence: Cannot predict future numbers, given the past
- True randomness can come from nature (thermal noise, background radiation from space)
  - Leveraging it is slow and expensive
  - Not easy to convert unpredictability to uniform distribution

Pseudo Random Number Generators (PRNG)

- Algorithmically generated random numbers (approximation of truly random)
- Linear Congruential Generators

  $X_{n+1} = (a * X_n + c) \bmod m$; X_0 is the seed
  - Certain choices of a, c and m can produce uniform distribution
- Not secure since attacker needs to be able to predict just m, a and c
- Better Variant: Take current clock time modulo m as a new seed; change seed after some numbers

Cryptographically Secure PRNG (CSPRNG)



Based on Cryptographic Primitives

- Many designs in literature
- Example: Two keys, two inputs and 3 EDE blocks
  - EDE: encrypt/decrypt/encrypt (2-key DES); can be replaced by AES
  - Next random number dictated by IV_{j+1} and one more independent pseudorandom input (time)
  - Cannot be predicted from Rj
    - Between two consecutive random numbers 3 EDE encryptions (nine DES encryptions)
  - IV_{j+1} cannot be predicted from IV_j; 2 EDE encryptions
- Much slower to generate pseudo-random numbers

True Random Number Generators (TRNG)

- Relies on analog phenomena (entropy source) that change with time in unpredictable ways and generates truly random stream
  - Entropy: extent of uncertainity with a random process; high good
  - E.g. amplify thermal noise in resistors and then digitize; Very power intensive
  - TRNG cannot generate at the rate applications need
- Leverage TRNG in CSPRNG. How?
  - Use TRNG as seeds to CSPRNGs

- Hardware: e.g. Intel digital random number generator (DRNG)
  - Uses invertors and thermal noise
- Software:
  - Timing between key presses
  - Mouse interrupt and position
  - Interrupt timings (disk better than timers)
  - Finishing time of block device requests
  - Few hundred bits of entropy per sec

Randomness in Linux

- Stores random data (see software) in a random pool
- Uses /dev/random (blocks) and /dev/urandom (non-blocking) to turn randomness into pseudo random numbers
- Any time random number generated □ entropy of randomness pool decreased
  - Entropy reaches zero /dev/random blocks
  - /dev/urandom will use data in pool as seed and generate as many pseudo random numbers as needed

References

- https://hackaday.com/2017/11/02/what-is-entropy-and-how-do-i-get-more-of-it/
- https://people.eecs.berkeley.edu/~daw/rnd/linux-rand
- https://people.eecs.berkeley.edu/~daw/rnd/