Chair for Network Architectures and Services
Institute of Informatics
TU München – Prof. Carle

Network Security

Chapter 2 Basics

2.4 Random Number Generation for Cryptographic Protocols

Motivation

- It is crucial to security that cryptographic keys are generated with a truly random or at least a pseudo-random generation process (see subsequently)
- Otherwise, an attacker might reproduce the key generation process and easily find the key used to secure a specific communication
- Generation of pseudo-random numbers is required in cryptographic protocols for the generation of
  - Cryptographic keys
  - Nonces (Numbers Used Once)
- Example usages
  - Key generation and peer authentication in IPSec and SSL
  - Authentication with challenge-response-mechanism, e.g. GSM and UMTS authentication

Random Number Generators

- Definition:

  A *random bit generator* is a device or algorithm which outputs a sequence of statistically independent and unbiased binary digits.

- Remark:
  - A random bit generator can be used to generate uniformly distributed random numbers
  - e.g. a random integer in the interval [0, n] can be obtained by generating a random bit sequence of length $\lfloor lg_2\ n \rfloor + 1$ and converting it into a number.
  - If the resulting integer exceeds *n* it can be discarded and the process is repeated until an integer in the desired range has been generated.

Entropy

(c.f. Niels Ferguson, Bruce Schneier: Practical Cryptography, pp. 155ff)

- The measure for „randomness“ is called „entropy“
- Let *X* a random variable which outputs a sequence of n bits
- The Shannon information entropy is defined by:

$$H(X) = -\sum_{x} P(X=x)\ln_2(P(X=x))$$

- E.g. if all possible outputs are equally probable, then

$$H(X) = -\sum_{i=0}^{2^n-1} \left(\frac{1}{2^n}\right)\ln_2\left(\frac{1}{2^n}\right) = -2^n * \frac{1}{2^n} * (-n) = n$$

- A secure cryptographic key of length *n* bits should have *n* bits of entropy.
- If *k* from the *n* bits become known to an attacker and the attacker has no information about the remaining (*n* – *k*) bits, then the key has an entropy of (*n*– *k*) bits
- A bits sequence of arbitrary large length that takes only 4 different values has only 2 bits of entropy
- Passwords that can be remembered by human beings have usually a much lower entropy than their length.
- Entropy can be understood as the average number of bits required to specify a bit-sequence if an ideal compression algorithm is used.

Pseudo-Random Number Generators (1)

- Definition:
  - A *pseudo-random bit generator (PRBG)* is a deterministic algorithm which, given a truly random binary sequence of length *k* (“seed”), outputs a binary sequence of length $m >> k$ which “appears” to be random.
  - The input to the PRBG is called the *seed* and the output is called a *pseudo-random bit sequence*.

- Remarks:
  - The output of a PRBG is not random, in fact the number of possible output sequences of length *m* with 2^k sequences is at most a small fraction of 2^m, as the PRBG produces always the same output sequence for one (fixed) seed
  - The motivation for using a PRBG is that it is generally too expensive to produce true random numbers of length *m*, e.g. by coin flipping, so just a smaller amount of random bits is produced and then a pseudo-random bit sequence is produced out of the *k* truly random bits
  - In order to gain confidence in the “randomness” of a pseudo-random sequence, statistical tests are conducted on the produced sequences

Pseudo-Random Number Generators (2)

- Example:
  - A linear congruential generator produces a pseudo-random sequence of numbers y_1, y_2, ... According to the linear recurrence

    $$y_i = a \times y_{i-1} + b \text{ MOD } q$$

    with a, b, q being parameters characterizing the PRBG
  - Unfortunately, this generator is predictable even when a, b and q are unknown, and should, therefore, not be used for cryptographic purposes

Random and Pseudo-Random Number Generation (3)

- Security requirements of PRBGs for use in cryptography:
  - As a minimum security requirement the length *k* of the seed to a PRBG should be large enough to make brute-force search over all seeds infeasible for an attacker
  - The output of a PRBG should be statistically indistinguishable from truly random sequences
  - The output bits should be unpredictable for an attacker with limited resources, if he does not know the seed
- <u>Definition:</u>

  A PRBG is said *to pass all polynomial-time statistical tests*, if no polynomial-time algorithm can correctly distinguish between an output sequence of the generator and a truly random sequence of the same length with probability significantly greater than 0.5

  - *Polynomial-time algorithm* means, that the running time of the algorithm is bound by a polynomial in the length *m* of the sequence

Random and Pseudo-Random Number Generation (4)

- Definition:
  - A PRBG is said *to pass the next-bit test*, if there is no polynomial-time algorithm which, on input of the first m bits of an output sequence s, can predict the $(m + 1)^{st}$ bit s_{m+1} of the output sequence with probability significantly greater than 0.5
- Theorem (universality of the next-bit test):

  A PRBG passes the next-bit test

  $\Leftrightarrow$

  it passes all polynomial-time statistical tests

  - For the proof, please see section 12.2 in [Sti95a]
- Definition:
  - A PRBG that passes the next-bit test – possibly under some plausible but unproved mathematical assumption such as the intractability of the factoring problem for large integers – is called a *cryptographically secure pseudo-random bit generator (CSPRBG)*

Hardware-Based Random Number Generation

- Hardware-based random bit generators are based on physical phenomena, as:
  - elapsed time between emission of particles during radioactive decay,
  - thermal noise from a semiconductor diode or resistor,
  - frequency instability of a free running oscillator,
  - the amount a metal insulator semiconductor capacitor is charged during a fixed period of time,
  - air turbulence within a sealed disk drive which causes random fluctuations in disk drive sector read latencies, and
  - sound from a microphone or video input from a camera
- A hardware-based random bit generator should ideally be enclosed in some tamper-resistant device and thus shielded from possible attackers

Software-Based Random Number Generation

- Software-based random bit generators, may be based upon processes as:
  - the system clock,
  - elapsed time between keystrokes or mouse movement,
  - content of input- / output buffers
  - user input, and
  - operating system values such as system load and network statistics
- Ideally, multiple sources of randomness should be “mixed”, e.g. by concatenating their values and computing a cryptographic hash value for the combined value, in order to avoid that an attacker might guess the random value
  - If, for example, only the system clock is used as a random source, than an attacker might guess random-numbers obtained from that source of randomness if he knows about when they were generated
- *Usually, such generators are used to initialize PRNGs, i.e. to set their seed.*

De-skewing

- Consider a random generator that produces biased but uncorrelated bits, e.g. it produces 1's with probability $p \neq 0.5$ and 0's with probability $1 - p$, where p is unknown but fixed
- The following technique can be used to obtain a random sequence that is uncorrelated and unbiased:
  - The output sequence of the generator is grouped into pairs of bits
  - All pairs 00 and 11 are discarded
  - For each pair 10 the unbiased generator produces a 1 and for each pair 01 it produces a 0
- Another practical (although not provable) de-skewing technique is to pass sequences whose bits are correlated or biased through a cryptographic hash function such as MD-5 or SHA-1

Statistical Tests for Random Numbers

- The following tests allow to check if a generated random or pseudo-random sequence inhibits certain statistical properties:
  - *Monobit Test:* Are there equally many 1’s as 0’s?
  - *Serial Test (Two-Bit Test):* Are there equally many 00-, 01-, 10-, 11-pairs?
  - *Runs Test:* Are the numbers of *runs* (sequences containing only either 0’s or 1’s) of various lengths as expected for random numbers?
  - *Autocorrelation Test:* Are there correlations between the sequence and (non-cyclic) shifted versions of it?
  - *Maurer’s Universal Test:* Can the sequence be compressed?
- The above descriptions just give the basic ideas of the tests. For a more detailed and mathematical treatment, please refer to sections 5.4.4 and 5.4.5 in [Men97a]

Examples for PRNGs

- Linear Congruential Generator
  - $X_{n+1} = (a\,X_n + b) \bmod m$
  - Very fast, but not suitable for cryptography!
- Suitable for cryptography
  - Blum Blum Shub
- On the basis of symmetric encryption
  - Output of block cipher in OFB or CTR mode
  - Output of a stream cipher (e.g. RC4)
    - Stream cipher = symmetric cipher that produces a random bitstream to be XORed with the plaintext
- On the basis of a cryptographic hash function
  - Iterate using hash function and seed, e.g.
    - X_0 = seed
    - $X_{i+1} = H\,(X_i \mid \text{seed})$

Addtional References

[Ferg03] Niels Ferguson, Bruce Schneier, „Practical Cryptography“, John Wiley & Sons, 2003

[Men97a] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press Series on Discrete Mathematics and Its Applications, Hardcover, 816 pages, CRC Press, 1997.

[Sti95a] D. R. Stinson. *Cryptography: Theory and Practice (Discrete Mathematics and Its Applications)*. Hardcover, 448 pages, CRC Press, 1995.