Technische Universität Braunschweig

Institute of Operating Systems and Computer Networks

Secure communication based on noisy input data

Entropy

Stephan Sigg

June 21, 2011

Overview and Structure

05.04.2011 Organisational
15.04.2011 Introduction
19.04.2011 Classification methods (Basic recognition, Bayesian, Non-parametric)
26.04.2011 Classification methods (Linear discriminant, Neural networks)
03.05.2011 Classification methods (Sequential, Stochastic)
10.05.2011 Feature extraction from audio data
17.05.2011 Feature extraction from the RF channel
24.05.2011 Fuzzy Commitment
31.05.2011 Fuzzy Extractors
07.06.2011 Error correcting codes
21.06.2011 Entropy
28.06.2011 Physically unclonable functions

Outline

Random sequences

Random number generation

Statistical tests

Conclusion

Random sequences

The security of many cryptographic systems depends upon the generation of unpredictable sequences

Random sequences

What is a random sequence?

Assume a sequence

$$U = U_0, U_1, U_2, \ldots$$

Let $0 \leq u_i < v_i \leq 1$ be real numbers

If U_i is a random variable, the probability that $u_i \leq U_i < v_i$ is equal to $v_i - u_i$

We say that U is a random sequence if

$$P[u_1 \leq U_i < v_1, \ldots, u_k \leq U_{n+k-1} < v_k] = (v_1 - u_1) \cdots (v_k - u_k)$$

for any value of k and for any choice of $0 \leq u_i < v_i \leq 1$[1]

[1] D.E. Knuth, The art of computer programming – seminumerical algorithms, Eddison-Wesley, 1998

Outline

Random sequences

Random number generation

Statistical tests

Conclusion

Random number generation

Definition

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

It can be used to generate uniformly distributed random numbers.

A random integer in the interval $[0, n]$ can be obtained by generating a random bit sequence of length $\lfloor \log n \rfloor + 1$ and interpreting it as a binary representation of an integer

For a truly random bit generator, some physical sources of random bits are used.

Random number generation

For common electronic devices, it is practically not feasible to employ a truly random bit generator.

In such cases, pseudo-random bit generators are utilised

Definition

A pseudo-random bit generator is a deterministic algorithm which, given a truly random binary sequence of length k, outputs a binary sequence of length $l \gg k$ which 'appears' to be random. The input to the PRBG is called the seed .

The output of a pseudo-random bit generator is not random.

Random number generation

The output of a pseudo-random bit generator is not random.

In particular, much fewer than 2^l distinct output sequences are possible

The security of such generators has to be verified by various statistical tests.

A minimum security requirement for pseudo-random bit generators is that the length of the random seed should not allow for complete enumeration of all possible values.

Random number generation

Example

A linear congruential generator generates a pseudo-random sequence of values $x_1, x_2, \ldots$ according to the linear recursive function

$$x_i = ax_{i-1} + b \mod m;$$

The parameters a, b, m characterise the generator while x_0 is the seed.

Such generators are commonly used for simulation purposes and probabilistic algorithms.

They are, however, predictable and hence not suitable for cryptographic applications

Note that these generators will pass common statistical tests!

Random number generation

A true random bit generator requires a naturally occurring source of randomness.

The generation of a hardware or software true RBG is therefore a difficult task.

Random number generation

Hardware-based generators

Hardware-based random bit generators build on the randomness occurring in physical sources.

These sources, however, might be biased, correlated or subject to adversarial observation.

- Elapsed time between emission of particles during radioactive decay
- Thermal noise from a semiconductor diode
- Frequency instability of an oscillator
- Sound from a microphone or video input
- noise fluctuation characteristic of an RF-channel

Random number generation

Software-based generators

Software-based random bit generator might build on processes such as

- System clock
- Elapsed time between keystrokes or mouse movements
- Content of input/output buffers
- user input
- system load or network statistics

The behaviour of these processes varies with the hardware platform utilised and also with environmental impacts

Possibly, an adversary might also

- obtain sufficient knowledge on these values in order to increase its chances to guess the secret correctly
- be able to manipulate values to increase its knowledge on the secret

Random number generation

De-skewing

A natural source of random bits may be defective in that the output bits may be biased or correlated

De-skewing is a technique to generate truly random sequences from such impaired bit sequences.

Example

Assume that a generator produces biased but uncorrelated bits with $P[x_i = 1] = p$ and $P[x_i = 0] = 1 - p$.
A process that divides the sequence of bits into a sequence of bit-pairs and interpretes any sequence 01 as 1 and 10 as 0 while discarding any sequence 11 and 00 results in an unbiased and uncorrelated sequence.

Random number generation

Pseudo-random bit generation

A one-way function f can be utilised to generate a pseudo-random bit sequence from a seed s by creating the sequence

$$f(s), f(s+1), \ldots$$

Depending on the one-way function it may be necessary, to discard some of the output values of the sequence in order to guard possible correlations between successive values.

Random number generation

ANSI X9.17 generator

FIPS-approved method for the purpose of pseudo-randomly generating keys and initialisation vectors for use with DES:

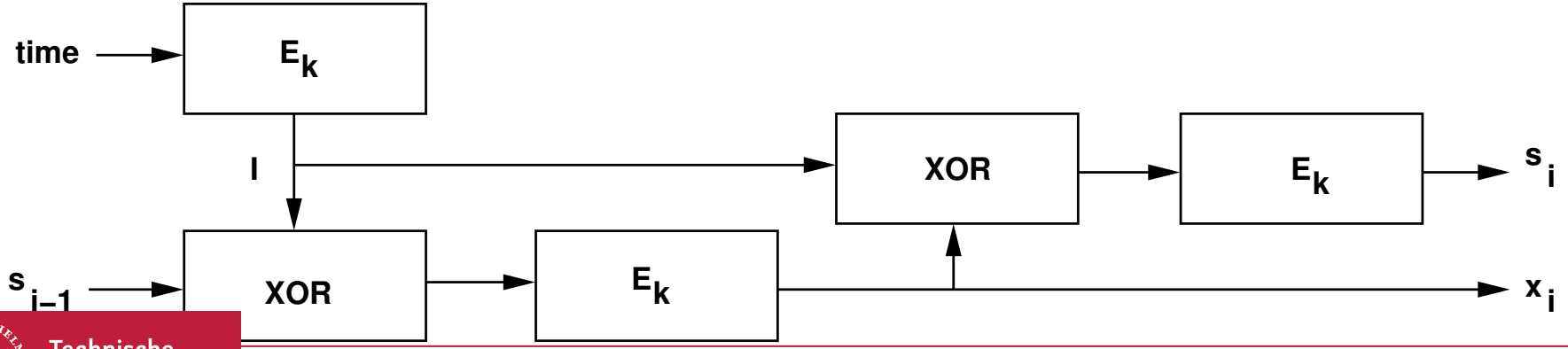
The PRBG is defined as

1: INPUT: 64-bit seed s_0, integer m, encryption key k,
2: OUTPUT: m pseudo-random 64-bit strings $x_1, x_2, \ldots, x_m$
3: Compute $I = E_k(date)$
4: **for** $i = 1..m$ **do**
5: $\quad x_i \leftarrow E_k(I \oplus s_{i-1})$
6: $\quad s_i \leftarrow E_k(x_i \oplus I)$
7: **end for**
8: **return** $(x_1, x_2, \ldots, x_m)$

Random number generation

ANSI X9.17 generator

1: INPUT: 64-bit seed s_0, integer m, encryption key k,
2: OUTPUT: m pseudo-random 64-bit strings $x_1, x_2, \ldots, x_m$
3: Compute $I = E_k(date)$
4: **for** $i = 1..m$ **do**
5: $\quad x_i \leftarrow E_k(I \oplus s_{i-1})$
6: $\quad s_i \leftarrow E_k(x_i \oplus I)$
7: **end for**
8: **return** $(x_1, x_2, \ldots, x_m)$



Random number generation

RSA pseudo-random bit generator

The RSA PRBG is a cryptographically secure pseudo-random bit generator under the assumption that the RSA problem is intractable.

1: OUTPUT: pseudo-random bit sequence $z_1, z_2, \ldots, z_l$
2: SETUP: Generate two secret RSA-like primes p, q
3: SETUP: Compute $n = pq$ and $\phi = (p-1)(q-1)$
4: SETUP: Select a random integer $e \in [1, \phi]$ such that $gcd(e\phi) = 1$
5: Select a random seed $x_0 \in [1, n-1]$
6: **for** $i = 1..l$ **do**
7: $\quad x_i \leftarrow x_{i-1}^e \mod n$
8: $\quad z_i \leftarrow$ least significant bit of x_i
9: **end for**
10: **return** $(z_1, z_2, \ldots, z_l)$

Outline

Random sequences

Random number generation

Statistical tests

Conclusion

Statistical tests

It is infeasible to give a mathematical proof that a generator indeed produces random bit strings

The reason is that for a purely random bit generator, any sequence of random bits is equally probable.

This means that such a generator might as well produce an arbitrary length 'conspicuous' sequence.

Example

For the 'game' Lotto (draw 6 numbers out of $[1, 49]$, the set $1, 2, 3, 4, 5, 6$ is as probably as any other arbitrary 6-number set.

Statistical tests

However, statistical tests are defined that can produce some evidence whether or not a generator might produce bit sequences that are considerably random

Each test determines whether a tested sequence contains some properties that a purely random sequence would contain only with small probability.

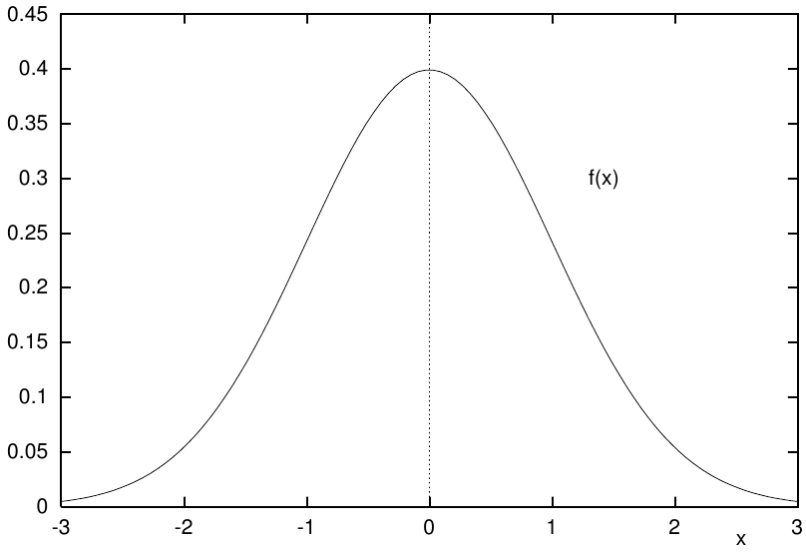
For instance, the number of 1-s and 0-s in the sequence should be roughly equal for most sequences

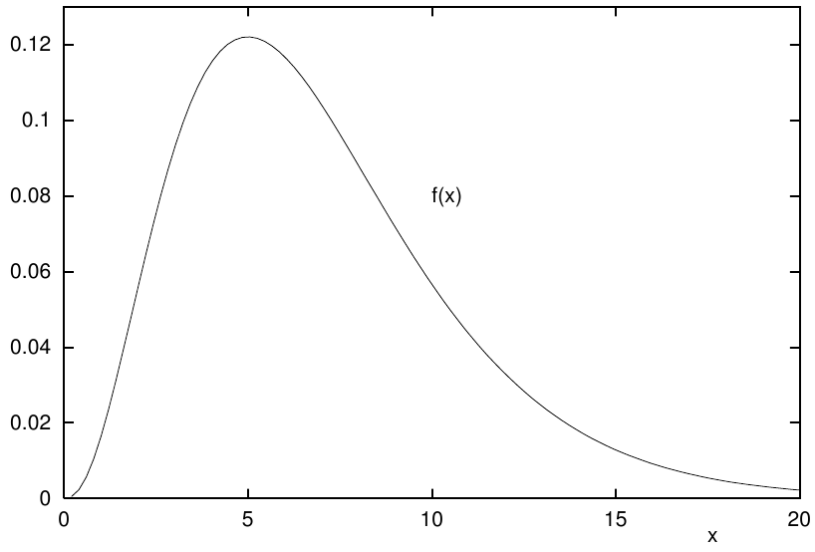
A statistical test can only be an indicator that a random number generator might produce strong random sequences or might have some weakness.

Statistical tests

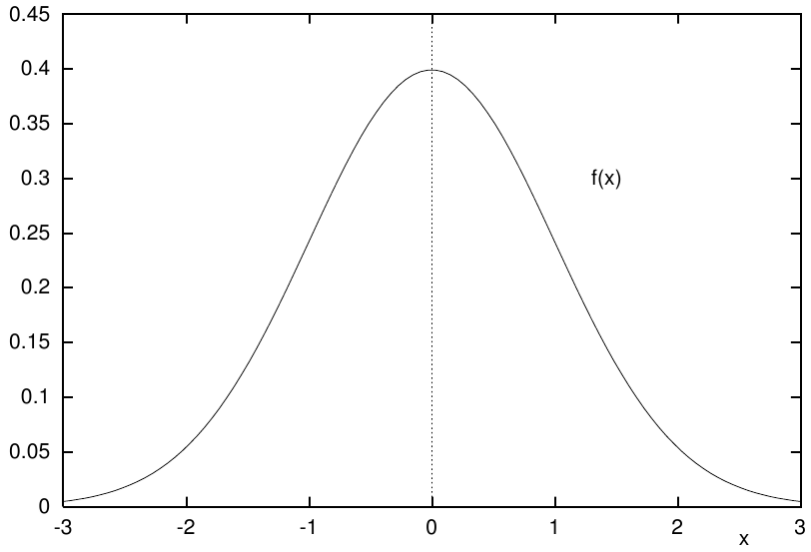
In order to test for the randomness of binary sequences, it is commonly checked if the property tested by a statistical test somehow follows a normal or chi-square distribution

The normal distribution arises when a large number of independent random variables with the same mean and variance are summed.





Statistical tests



A continuous random variable X has a normal distribution with mean μ and variance σ^2 when its probability density function is defined by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(-x-\mu)^2}{2\sigma^2}}, -\infty < x < \infty$$

Statistical tests

In order to test for the randomness of binary sequences, it is commonly checked if the property tested by a statistical test somehow follows a normal or chi-square distribution

Let $v \geq 1$ be an integer. A continuous random variable X has a χ^2 distribution with v degrees of freedom if its probability density function is defined by

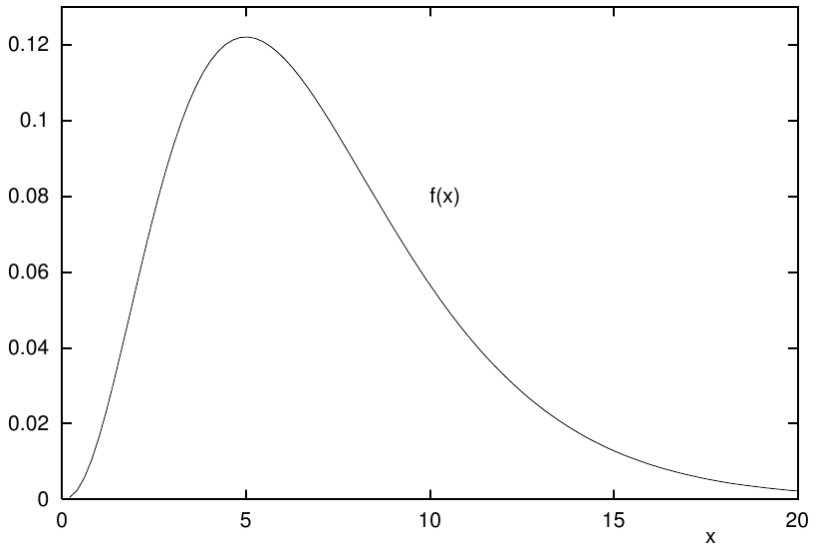
$$f(x) = \begin{cases} \frac{1}{\Gamma(\frac{v}{2})2^{\frac{v}{2}}} x^{(\frac{v}{2})^{-1}} e^{-\frac{x}{2}} & , 0 \leq x < \infty, \\ 0 & , x < 0 \end{cases}$$

Where Γ is defined by

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx, \text{ for } t > 0$$

Statistical tests

In order to test for the randomness of binary sequences, it is commonly checked if the property tested by a statistical test somehow follows a normal or chi-square distribution



Statistical tests

Frequency test (monobit test)

The purpose of this test is to determine whether the number of 0-s and 1-s are approximately equal in a random sequence

Let n_0, n_1 denote the number of 0-s and 1-s in a random sequence

The statistic used is

$$X_{\text{Monobit}} = \frac{(n_0 - n_1)^2}{n}$$

This approximately follows for $n \geq 10$ a χ^2-distribution with 1 degree of freedom.

(typically large values of $n \gg 10000$ are utilised)

Statistical tests

Serial test (two-bit test)

The purpose of this test is to determine whether the number of occurrences of $00, 11, 10, 01$ are approximately the same.

Let $n_0, n_1, n_{00}, n_{11}, n_{10}, n_{01}$ denote the number of occurrences of the sequences $0, 1, 00, 11.10, 01$

The statistic used is

$$X_{\text{Serial}} = \frac{4}{n-1}\left(n_{00}^2 + n_{11}^2 + n_{10}^2 + n_{01}^2\right) - \frac{2}{n}\left(n_0^2 + n_1^2\right) + 1$$

This approximately follows a χ^2 distribution with 2 degrees of freedom for $n \geq 21$

Statistical tests

Runs test

The purpose of this test is to determine whether the number of 1-sequences and 0-sequences (runs) of various lengths is as expected for a random sequence.

The expected number of 0/1-sequences of length i can be estimated for a random sequence of length n by

$$e_i = \frac{n - i + 3}{2^{i+2}}$$

Let k be the largest integer for which $e_k \geq 5$.

Let S_0, S_1 denote the number of 0 or 1-sequences of length i for $1 \leq i \leq k$.

Statistical tests

Runs test

Let S_0, S_1 denote the number of 0 or 1-sequences of length i for $1 \leq i \leq k$.

The statistic of this test is

$$X_{\text{Runs}} = \sum_{i=1}^{k} \frac{(S_1 - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(S_0 - e_i)^2}{e_i}$$

This equation should approximately follow a χ^2 distribution with $2k - 2$ degrees of freedom.

Statistical tests

Autocorrelation test

The number of bits in a sequence s that are not equal to their d-shifts is

$$A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$$

The statistic used is

$$X_{\text{Autocorrelation}} = 2\frac{\left(A(d) - \frac{n-d}{2}\right)}{\sqrt{n-d}}$$

This approximately follows a normal distribution for $n - d \geq 10$.

Statistical tests

Test suites

- FIPS 140-2 statistical tests for randomness[2] (Federal Information Processing Standards)
- Die Hard battery of statistical tests [3] (George Marsaglia, Florida State University)
- Die Harder [4] (Robert G. Brown)
- NIST statistical tests (FIPS 140-1) [5] (National Institute of Standards and Technologies)

[2] http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
[3] http://www.stat.fsu.edu/pub/diehard/
[4] http://www.phy.duke.edu/~rgb/General/dieharder.php
[5] http://csrc.nist.gov/groups/ST/toolkit/rng/documents/sts-2.1.zip

Statistical tests

The null hypothesis

The null hypothesis for random number generator testing says, that a generator is a perfect random number generator and for any choice of seed produces an infinitely long, unique sequence of numbers that have all the statistical properties of random numbers

This assumption can not hold for any pseudo-random number generator

However, a good pseudo-random number generator should be for practical applications sufficiently close to fulfilling this hypothesis

Statistical tests

The null hypothesis

In order to test the null hypothesis, the random number generator is utilised to generate a sequence of presumably random numbers

These numbers are applied to a wide range of test statistics.

From a knowledge of the target distribution we can then estimate the probability that the sequence is truly random

This probability is called the *p*-value for a particular test-run

Statistical tests

The null hypothesis

For a good pseudo-random number generator the p-values should be approximately evenly distributed in $[0, 1]$

This means that e.g. a random number generator should produce p-values less than 0.05 in 5% of all cases

Outline

Random sequences

Random number generation

Statistical tests

Conclusion

Questions?

Stephan Sigg
sigg@ibr.cs.tu-bs.de

Literature

C.M. Bishop: Pattern recognition and machine learning, Springer, 2007.

P. Tulys, B. Skoric, T. Kevenaar: Security with Noisy Data – On private biometrics, secure key storage and anti-counterfeiting, Springer, 2007.

W.W.Peterson, E.J. Weldon, Error-Correcting Codes, MIT press, 1972.

R.O. Duda, P.E. Hart, D.G. Stork: Pattern Classification, Wiley, 2001.