0368.4162: Introduction to Cryptography Ran Canetti

Lecture 3

17 November 2008 Fall 2008

Scribes: Meir-Johnathan Dahan & Tomer Levinboim

Topics

1. Existence of weak one-way functions imply existence of strong ones
2. Stream Ciphers
3. Pseudo-Random-Generators (PRG) preliminaries

Preliminaries

Definition 1: (Uniform Binary Distribution)

Let U_n denote the *uniform binary distribution* over the $\{0,1\}^n$ domain.

Definition 2: (Negligible Function)

a function $\nu : N \to R$ is said to be *negligible* if $\forall c \in N$ there exists an n_0 such that we have $\nu(n) < n^{-c}$ for all $n > n_0$

Definition 3: (Efficient Algorithm)

A non-uniform polynomial-time in its input length (restricted to language recognition, these algorithms correspond to the P/poly complexity class which is known to contain BPP).

Also recall from the previous lecture:

Definition 4: (One-Way Function)

$f : \{0,1\}^* \to \{0,1\}^*$ is a (strong) *one-way function* if:

1. f is efficiently computable.
2. f is not invertible by any efficient algorithm on "almost" all the range of f. formally, for all non-uniform polynomial algorithms A, there exists a negligible function ν such that for any input length n:

$$\Pr_{x \leftarrow U_n}[f(A(1^n, f(x)) = f(x)] < \nu(n)$$

That is, the probability to invert f is negligible.

Defintion 5: (Weak One-Way Function)

$f : \{0,1\}^* \to \{0,1\}^*$ is a *weak one-way function* if:

1. f is computable in polynomial time.
2. f is not invertible by a polynomial algorithm on at least a non-negligible part of the range, or formally: There exists a constant $c > 0$ such that for all efficient algorithms A:

$$\Pr_{x \leftarrow U_n}[f(A(1^n, f(x)) = f(x)] < 1 - n^{-c}$$

That is, it is easy to invert f on almost all of its range but a negligible part.

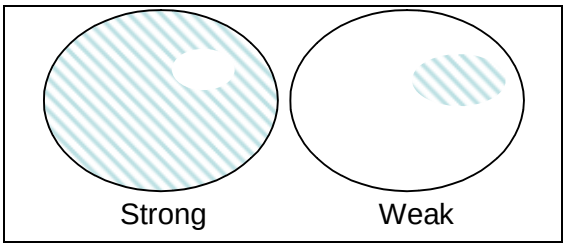

Image 1: (Strong) one-way function vs. weak one-way function.
The clear part represents easy to invert images while the filled part represents hard ones.

Hardness Amplification

Generally it is not known whether one-way functions even exist, however we will now show a construction of a strong one-way function given a weak one-way function. This result is called *hardness amplification* - intuitively given such a weak one-way function f, we shall apply it enough times such that at least one of its inputs will be hard to invert.

Theorem 1: (Existence of weak one-way function, implies existence of strong ones)
Let $f:\{0,1\}^n \to \{0,1\}^n$ be a weak one-way function, then there exists a $t \in \mathbb{N}$ polynomial in the input length n, such that $f':\{\{0,1\}^n\}^t \to \{\{0,1\}^n\}^t$ as defined below is a strong one-way function:

$$f'(x_1, x_2, ..x_t) = (f(x_1), f(x_2)..., f(x_t))$$

Notice that each $x_i \in \{0,1\}^n$

If we can show that for almost all $\vec{x} = (x_1, x_2, ..x_n)$, at least one x_j belongs to the "hard" part of f, then we can conclude that f' is strongly one-way.

Notice that if we take $t = n^c$ we can expect to have one non-invertible x_j (for some j), in the proof however, we use $t = n^{c+1}$ in order to increase the probability of this event.

The above discussion is good for intuition, but does not provide a proof as, a theoretic inverting algorithm A need not necessarily try to invert each $f(x_j)$ independently on its own, but may try to exploit the dependencies between the different values of the $f(x_j)$'s. (recall the discussion about this "wrong proof" from the previous lecture, and notice that in general we should not assume anything about the manner in which an efficient algorithm works, unless we know nothing is lost by such an assumption) Instead we will proceed to prove by reducing the hardness of f' to that of f (a typical proof structure in cryptography).

Proof (Theorem 1)

We will prove by contradiction by assuming f' is not a strong one-way function.

Contradicting the strong "one-wayness" of f', let A' be a poly-time algorithm that inverts f' with probability $\geq n^{-\hat{c}}$ for some $\hat{c} > 0$.

Given any $c > 0$, we will use A' to construct an efficient algorithm A that inverts f with probability $> 1 - n^{-c}$, thereby contradicting f being a weak one-way function (where c is the constant taken from the hardness definition of a weak one-way functions).

Define the following procedure I_0 as a single trial for inverting y:

I_0

Input: $1^n, y$ (an output of f we wish to invert)

Algorithm:

For $j = 1..t$

1. Choose $x_1..x_t \in U_n$
2. Compute $(z_1,..z_t) \leftarrow A'[f(x_1), f(x_2)..., f(x_{j-1}), y, f(x_{j+1})..., f(x_t)]$
3. If $y = f(z_j)$ output z_j and halt

To improve our chances of inverting f, we will run I_0 multiple times. That is, define the algorithm A to run I_0 n^d times with its input y for some d (whose value depends on c and $\hat{c}$, to be defined later on), outputting the first result that matches the criterion in step 3. If after n^d no appropriate z_j is found, output FAIL.

Remark: we could have defined I_0 to pick j uniformly at random instead of incrementing, however the current definition simplifies the analysis.

We now wish to show that A succeeds almost always, even though A' may rarely succeed. Given this define $G \subseteq \{0,1\}^n$ - informally, the set of good inputs x which are easy to invert by I_0 - as follows:

$$G = \{x \mid \Pr_{I_0}[I_0 \text{ inverts } f(x)] > n \cdot \frac{1}{n^d}\}$$

We would like to show that G upholds two properties:

1. For each $x \in G$, A inverts $f(x)$ with a very probability $\geq 1 - e^{-n}$

   Claim 1: $\forall x \in G: \Pr_A[A \text{ inverts } f(x)] > 1 - e^{-n}$

   Note: here the subscript A refers to the random coins A uses.

2. Almost all input in the domain $\{0,1\}^n$ are in G

   Claim 2: $\frac{|G|}{2^n} \geq 1 - \frac{1}{2n^c}$

Recall: (Marginalization of Conditional Probabilities)
For any event $A \subseteq \Omega$

$$P(A) = P(A,B) + P(A,\overline{B}) = P(A \mid B)P(B) + P(A \mid \overline{B})P(\overline{B}) \quad (3.1)$$

The proof of Theorem 1 will follow from these two claims as:

$$\begin{aligned} &\Pr_{A,x}[A \text{ fails to invert } f(x)] \\ &= \Pr[A \text{ fails to invert } f(x) \mid x \in G]\Pr[x \in G] \\ &+ \Pr[A \text{ fails to invert } f(x) \mid x \notin G]\Pr[x \notin G] \\ &\leq e^{-n} + \frac{1}{2n^c} \leq n^{-c} \end{aligned}$$

Where the first equality is due to (3.1), and the second inequality is due to claim 1 applied to $\Pr[F_{A,x} \mid x \in G]$ and claim 2 applied to $\Pr[x \notin G]$, and then upper-bounding the other terms by 1.
This contradicts f being a weak one-way function, as it is easy to invert on almost all of its range aside of a less than negligible part (For example, consider a function that is hard to invert on only a 2^{-n} part of the range. It clearly doesn't meet the weak one-way function definition. The existence of a polynomial fraction of hard to invert points is necessary for the proof to go through. We do not know of a hardness amplification that can start with a negligible fraction of hard points).

We are now left with proving Claim 1 and Claim 2:

Proof of Claim 1: $(\forall x \in G : \Pr_A[A \text{ inverts } f(x)] > 1 - e^{-n})$

The adversary/algorithm A repeats I_0 n^d times. The attempts are independent (since x is fixed, so each iteration depends only on the random coins of A), each having success probability n/n^d at least, and G is defined for a single iteration of I_0. Therefore:

$$\Pr_A[A \text{ fails inverting } f(x) \text{ on all } n^d \text{ iterations} | x \in G] \leq (1 - \frac{n}{n^d})^{n^d} = \frac{1}{e^n}$$

And therefore,

$$\Pr_A[A \text{ inverts } f(x) | x \in G] \geq 1 - \frac{1}{e^n}$$

Proof of Claim 2: (G is large or $|G| \geq 2^n - 2^n / 2n^c$),
Let S be the event that A' inverts $f'(x_1,...,x_t)$ and we partition S to two disjoint events:

1. $S_1 = S \wedge (\exists j : x_j \notin G)$ - that is, A' inverted although one of the $f(x_j)$'s is hard to invert.
2. $S_2 = S \wedge (\forall j : x_j \in G)$ - all $f(x_j)$'s are (relatively) easy to invert.

Clearly,

$$\Pr[S] = \Pr[S_1] + \Pr[S_2] \quad (3.2)$$

Recalling our hypothesis, that f' is not a strong one-way function, and that A' in particular is the algorithm that successfully shows this, we know that:

$$\Pr[S] \geq (tn)^{-\hat{c}} = n^{-c\cdot\hat{c}-2\hat{c}} \qquad (t=n^{c+1})$$

Intuitively, S_1 is unlikely, as inverting an $x \notin G$ is unlikely, so event S_2 must be likely, but for that to happen, G must be large. More precisely we'll show:

Lemma: $\Pr[S_1] \leq n^{-(d-c-2)}$

Which will imply that S_2 is very large.

Deferring the proof of the lemma we obtain from it and (3.2) that:

$$\Pr[S_2] \geq n^{-c\cdot\hat{c}-2\hat{c}} - n^{-(d-c-2)},$$

We now finally choose $d = c\cdot\hat{c} + 2\hat{c} + c + 3$ to obtain (recall d controls the number of iterations)

$$\Pr[S_2] \geq n^{-\hat{c}\cdot c-2\hat{c}} - n^{-\hat{c}\cdot c-2\hat{c}-1} \geq \frac{1}{2n^{\hat{c}\cdot c+2\hat{c}}} \quad (3.3)$$

However, if we assume Claim 2 is wrong (formally: $\Pr_x[x \in G] < 1 - 1/2n^c$) we obtain:

$$\begin{aligned}\Pr[S_2] &= \Pr[S \wedge (\forall j : x_j \in G)] \\ &\leq \Pr[(\forall j : x_j \in G)] \\ &\leq (1 - 1/2n^c)^{n^{c+1}} \\ &\leq (1/e)^n \leq 1/poly(n)\end{aligned}$$

Contradicting (3.3).
We now move on to prove the lemma, which will complete the proof of claim 2.

Proof of Lemma: ($\Pr[S_1] \leq n^{-(d-c-2)}$)
First notice that for all $a \notin G$, $i \in [t]$:

$$\Pr_{A,\vec{x}}[A' \text{ inverts } f'(\vec{x}) \mid x_i = a] \leq \Pr_{I_0}[I_0 \text{ inverts } f(a)] \leq \frac{n}{n^d} \qquad (3.4)$$

Where the first inequality holds since in I_0 iterates over all $j \in [t]$, and the second inequality follows from the definition of G.

Now:

$$\Pr_x[S_1] = \Pr[S \wedge (\exists j : x_j \notin G)]$$

$$(1) \qquad \leq \sum_{j=1}^{t} \sum_{a \notin G} \Pr[S \wedge x_j = a]$$

$$(2) \qquad = \sum_{j=1}^{t} \sum_{a \notin G} \Pr[S \mid x_j = a] \Pr[x_j = a]$$

$$(3) \qquad \leq \sum_{j=1}^{t} \max_{a \notin g} \Pr[S \mid x_j = a]$$

$$(4) \qquad \leq \sum_{j=1}^{t} \frac{n}{n^d} = t \frac{n}{n^d} = n^{c+1} \frac{n}{n^d} = n^{-(d-c-2)}$$

(1) is due to union bound (2) is due to conditional probability, (3) is justified by noticing that we are substituting a weighted average, by a maximum function and finally, (4) is by (3.4).
Thus we have completed the proof of the lemma and Claim 2 in turn.

□

The theorem shows that the existence of weak one-way functions is equivalent to that of strong one-way functions.

Although we have shown how to construct a strong one-way function f' from a weak one, we comment that this construction is not very good, in the sense that we would expect more interference in the computation derived from the inputs, while we have not done so.

Moreover, the result we have obtained seems weak if we compare the computational strength of adversaries that can invert f to those that can invert f' - our construction yields a strong one-way function which is (sadly) only secure against far weaker adversaries, compared to the weak one-way function we used to construct it. More formally, if f is secure against adversaries that run in $q(n)$-time, A' is secure against adversaries even weaker than $q(n)n^{-d}$.
To get an intuition about this, consider the fact that we have assumed the existence of a "black-box" algorithm A' which efficiently inverts a strong one-way function with some running time (say, $\Theta(p(n))$), only to obtain a new algorithm A whose running time is far worse in its input length n, compared to A'. Explicitly, if indeed A' runs in time $\Theta(p(n))$ in its input length, I_o runs in time $\Theta(t \cdot p(tn))$, and therefore A runs in $\Theta(n^d \cdot t \cdot p(tn))$ time.
This may stem from the treatment of A' as a black box rather than actually inspecting it closely and using the insight that allows it to invert f' efficiently.

One Time Pad and Stream Ciphers

We now move on to define and later on construct another cryptographic primitive, namely the *Pseudo-Random Generators*. We start with a motivating discussion:

Consider the following problem: Alice (denote A) wishes to send a message to Bob (denote B) over an unsecured channel. For the purpose of secrecy, A wishes to encrypt the message before sending to B and we assume that A and B can agree on a mutual secret key beforehand.
Assuming they don't care about the secret key's length, there is a perfect solution for this problem called the One-Time-Pad (OTP) encryption:
Suppose A and B share a long secret s of random bits that are not available to eavesdroppers, then secure communication is easy - Let A's message m be composed of n bits $m = (m_1, ...m_n)$, A obtains n bits $r = (r_1, ...r_n)$ from the "One Time Pad" s and proceeds to compute the ciphertext c as follows (where $\oplus$ denotes the bitwise-XOR operator):

$$c = (c_1, ...c_n) = (m_1 \oplus r_1, m_2 \oplus, ...m_n \oplus r_n)$$

To decrypt the message, B simply uses the agreed upon r to obtain $m = c \otimes r$. If indeed s is statistically random then so is c (independently of m!) and an eavesdropper E will be unable to obtain even a single bit of m regardless of its computational power. Thus, it is easy to see that the OTP is "perfectly secret", a term we haven't formalized yet, but never the less it is clear that as long as the bits are random and unknown the produced ciphertext c is completely independent from the plaintext m.
In other words, for any two distinct messages $m, m' \in \{0,1\}^n$ we have that the random variables $c = m \oplus r$ and $c' = m' \oplus r$ are distributed identically.
Notice that the used r bits should be discarded since if E obtains $m \oplus r$ and $m' \oplus r$ she can compute $m \oplus m'$ which is not uniform and therefore reveals information about m and m'.

Although the OTP provides perfect secrecy, it is not used in practice as generating truly random bits is a hard task, but mostly, because its users need to predict the length of all of their future communication and agree on a with at least the same length.
To overcome this issue, A and B will surely need to use a short secret key, which as mentioned on the first lesson, will not provide them with perfect secrecy (Shannon). Cryptography suggests using Pseudo-Random Generators (PRG) – functions that work on a seed (key) of length k and produce a "random-looking" string of length n>k.

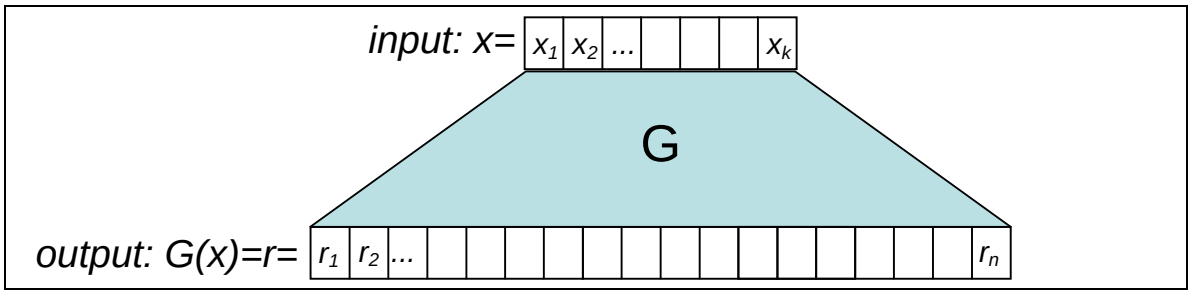

Image 2: The Pseudo-Random Generator G receives a seed x of length k
and produces a random string of length n>k

Using this output we can maintain the same overall structure of the encryption scheme – generating a long "random looking" pad to be xored with the message. The term "stream cipher" refers to this structure and a PRG is an abstraction which specifies the requirements from a stream cipher.

Example: LFSR (Linear Feedback Shift Register)
Consider the following deterministic algorithm G which takes an input seed of size k and outputs a long string of "random-looking" bits
Given an initial seed of size n bits, the LFSR generates a stream of bits that appear to be random. But it is impossible to create an endless stream of random data from a limited amount of information.

The construction of LFSR is as follows:

- Let $S = (s_k, s_{k-1}, ...s_1)$ be our initial key of size k bits
- Let $B = (b_k, b_{k-1}, ...b_1)$ be our feed back register of size k bit.
- Let $I \subset [n]$ of indexes $I = \{i_1, i_2, ...i_t\}$

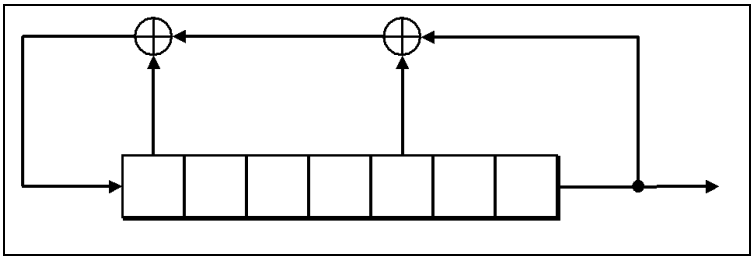
Image 3: an LFSR circuit for generating a "random looking" string
with parameters $k = 7, I = \{3, 7\}$

Algorithm (Sketch):
Initially assign B the value of the secret S:

$$B \leftarrow S$$

To produce a new "random looking" bit b, we XOR the bits at the indices indicated in I to produce a new bit b as follows:

$$b = \bigoplus_{i \in I} B_i$$

On each iteration, the entire register shifts one bit to the right and the bit at position 1 is extract as the random bit. The new bit b is inserted as the left-most bit.
In our example above (Image 2), b depends on the values of b_3, b_7 and b_1, and indeed different LFSR implementations differ in the seed/register length k and the content of the index set I.

Clearly LFSR does not produce truly random bits as, the cipher will start to repeat itself within less then 2^k steps (the maximum number of possible configuration of B). It is also not clear if all 2^k values will be exhausted, and it might be the case that the LFSR will get "stuck" in a shorter period. Research as shown that it is possible to ensure a long period by having the index set I encode an irreducible polynomial over $GF(2^k)$.

In itself this scheme is not effective as given k consecutive bits of its output, it is possible to know the internal state of the LFSR and thus predict the following bits (in particular, the first k bits are the key). To circumvent this problem, researchers suggest to add an additional (usually non-linear) operation on the final stage before outputting.

One such approach for strengthening LFSR for usage as a PRG is the Shrinking Generator [Coppersmith, Krawczyk, Mansour 93]:

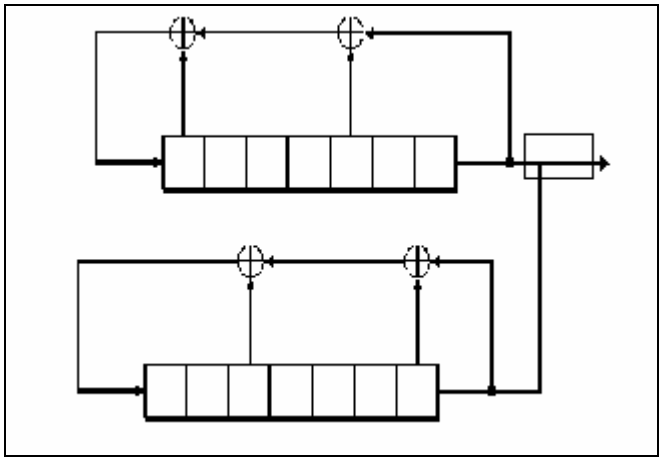

Image 4: a Shrinking Generator; in the final step, the first bit of the upper register will be used as a random bit only when the lower register's first bit is 1.

Given a seed (key) S split its bits between the two LFSR's L_1 and L_2. Run both L_1 and L_2 at the lockstep and output L_1's bit b_1^1 as the stream cipher if and only if L_2's output bit b_1^2 is 1, otherwise discard the bit. This breaks the linearity of the generator making it less predictable. In fact, there are no known weaknesses for this cipher.

Another well-known stream cipher is RC4, designed by Ron Rivest and widely used in practice (e.g., WEP – an algorithm for securing wireless networks).

Computational Indistinguishability

The former discussion was strictly intuitive, as we haven't defined what a "random looking" string is, and in what sense do we obtain secrecy. We will now proceed to formalize our discussion and introduce several tools and definitions that will aid us define these notions of randomness and secrecy.

Definition: (Statistical Distance I)

Let D_1, D_2 be two distributions over the same domain D and let $f : D \to \{0,1\}$ then the *statistical distance* δ_f between D_1, D_2 with respect to the function f is defined as:

$$\delta_f(D_1, D_2) = |\Pr_{d \in D_1}[f(d) = 1] - \Pr_{d \in D_2}[f(d) = 1]|$$

D_1, D_2 are called $\varepsilon - statistically - close$ if $\max_{f:D \to \{0,1\}}[\delta_f(D_1, D_2)] < \varepsilon$

Alternatively, we write $SD(D_1, D_2) < \varepsilon$.

An alternative, but equivalent definition for statistical difference is the following:

Definition: (Statistical Distance II)

Let D_1, D_2 be two distributions over the same domain D and let Δ be their common support then the *statistical distance* δ between D_1, D_2 is defined as:

$$\delta(D_1, D_2) = \frac{1}{2}\sum_{\delta \in \Delta} |\Pr_{d \leftarrow D_1}[d = \delta] - \Pr_{d \leftarrow D_2}[d = \delta]|$$

Formulation II is the common definition of statistical distance between two distributions, while formulation I is used to motivate the notion of computational Indistinguishability. Indeed, using formulation I it is straight forward to define the computational analogue, given our experience with one-way functions:

Definition: (Computational Distance)

Given the above definitions and an algorithm A then the *computational-difference* δ_A between D_1, D_2 with respect to A is defined as:

$$\delta_A(D_1, D_2) = \frac{1}{2}|\Pr_{d \in D_1}[A(d) = 1] - \Pr_{d \in D_2}[A(d) = 1]|$$

D_1, D_2 are called $\varepsilon(t) - computationally - close$ if $\max_{A \text{ runs in time t}} \delta_A(D_1, D_2) < \varepsilon(t)$

Moving on to deal with asymptotic "closeness" of distribution (as usual, for simplicity, flexibility and generality), we shall start with the notion of probability ensembles:

Definition: (Probability Ensemble)
A probability ensemble is an infinite sequence of distributions

$$\hat{D} = \{D_n\}_{n\in\square}$$

where each D_n is a probability distribution.
We will usually think of D_n as describing an experiment with security parameter dependent on n.

Now we can define the statistical and computational distance in asymptotic terms:

Definition: (Statistical Distance for Distribution Ensemble)
Two distribution ensembles $\hat{D}^1, \hat{D}^2$ are *statistically-close*, denoted $\hat{D}^1 \approx_s \hat{D}^2$

$$SD(D^1{}_n, D^2{}_n) = \max_f[\delta_f(D^1{}_n, D^2{}_n)] \le \nu(n)$$

for a negligible function ν.

Definition: (Computational Distance for Distribution Ensemble)
Similarly to the above definitions, two distribution ensembles $\hat{D}^1, \hat{D}^2$ are *computationally-close*, denoted $\hat{D}^1 \approx_c \hat{D}^2$ if for any non-uniform poly-time algorithm A (also known as a distinguisher), the distance

$$\delta_A(D^1{}_n, D^2{}_n) \le \nu(n)$$

for a negligible function ν.

Technical remarks:

1. This is weaker than saying that for any n (or even for any large enough n) the distributions $D^1{}_n, D^2{}_n$ are $\varepsilon(t)$-close where ε is a negligible function.
2. A should be given 1^n as input in addition to the samples from $D^1{}_n$ and $D^2{}_n$

Finally, lets us define what it means for a distribution (ensemble) to be a Pseudo-Random

Definition: (Pseudo-Random)
An ensemble $\hat{D}$ is *Pseudo-Random* if $(\hat{D}, \hat{U})$ are computationally close, where $\hat{U} = \{U_n\}_{n\in N}$ and for all n U_n is the uniform distribution $\{0,1\}^n$.

The concept of computational closeness is a "deep" concept, it is a basis for a new statistics notion – whatever cannot be done by an efficient computation effectively, cannot be done at all. It trades off two seemingly unrelated concepts: Computational hardness and Randomness. This mixing of the two concepts underlies most of cryptography and in fact, it has also trickled into many areas in computer science.

References

1. Computational Complexity: A Conceptual Perspective by Oded Goldreich Chapter 7 / Section 7.1.2 – Amplification of Weak One-Way Functions.
2. Computational Complexity: A Modern Approach by Sanjeev Arora and Boaz Barak; Chapter 10 – Cryptography;
   (Draft: http://www.cs.princeton.edu/theory/complexity/)