T-79.511 Special Course on Cryptology / Provable Security: Relations between OWFs, PRGs and PRFs (revised)

Jan-Erik Ekberg

February 19, 2002

1 Introduction

These lecture notes intend to give an overview of the provable relations between one-way functions (OWFs), pseudorandom generators (PRGs) and pseudorandom functions (PRFs). Only a selected number of proofs and reductions are thoroughly presented in this paper (most are left to the references), but the selection hopefully gives the reader some notion of the way the random oracle model can be used for proving theorems. The reader should be warned that the constructs throughout the paper are based on *function ensembles* (definition later) in spite of the notation which is the same as is use for functions. In the few cases functions are constructed as parts of proofs the distinction should be evident from the setting.

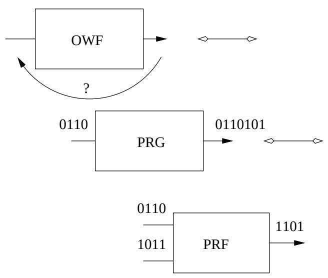

Figure 1: *OWF-PRG-PRF*

We start out by examining the relation between OWFs and PRGs while getting acquainted to some supporting theorems and methods. The paper the continues by an examination of the much more straight-forward constructs between PRGs and PRFs. Conclusions and references follow.

2 OWF

A one-way function is intuitively a function that is hard to invert according to some given measure. In the following definition we define *hard* based on the work load necessary for an adversary to find out the input, given the output of the function.

Definition 2.1 Let $f : \{0,1\}^{t_n} \to \{0,1\}^{l_n}$ be a P-time function ensemble, and let $X \in_U \{0,1\}^{t_n}$. The success probability of the adversary A for inverting f is $sp_n(A) = Pr[f(a(f(X))) = f(X)]$. Function ensemble f is a R-secure one-way function if there is no R_{t_n}-breaking adversary for f.

This definition needs some supporting knowledge to be understandable. A P-time function ensemble requires that the function is computable in polynomial time with respect to the number of input bits. An R_{t_n}-breaking adversary will break (in this case invert) f in polynomial time $(O(P(t_n)))$ with respect to the maximum running time T_n. Including success probability in the calculation gives $R_{t_n} = T_n/sp_n(A)$. What we ideally are looking for is a OWF that has exponential $R_{t_n} = O(t_n) \cdot 2^{t_n}$ - breaking time, i,e. it is as hard to break as trying out all possible inputs of f. X is a uniformly distributed random variable over the input $\{0,1\}^{t_n}$.

3 PRG

In order for us to define a pseudorandom generator we take into use the term 'computationally indistinguishable'. In practice this means that any adversary (A) cannot tell the difference between samples drawn from two 'matching' random distributions (in this case our pseudorandom computational source is one, and a variable drawn from a uniform random distribution the other). Formally, for random distributions X and Y drawn from probability ensembles $D : \{0,1\}^{l_n}$ and $E : \{0,1\}^{l_n}$ we define the success probability for the adversary $sp_n(A) = |Pr[A(X) = 1] - Pr[A(Y) = 1]|$ and state that D and E are R_{l_n} computationally indistinguishable if there is no R_{l_n}-breaking adversary for distinguishing them. An alternative definition is to use $sp1_n(A) = (sp_n(A))^2$ because it more accurately reflects the trade-off between the running time of A and its success probability [5].

Now we can proceed to define a pseudorandom generator as

Definition 3.1 Let $f : \{0,1\}^{t_n} \to \{0,1\}^{l_n}$ be a P-time function ensemble where $l_n > t_n$. f is a R_{l_n}-secure pseudorandom generator if the function (probability) ensembles $f(U_{t_n})$ and U_{l_n} are R_{l_n}-secure computationally indistinguishable.

Here we must note that the output length is defined to be longer than the input length, meaning that a pseudorandom generator must stretch its input by at least 1 bit. A construct (proven in [3]) can stretch the PRG further (and this is later needed when we look into the connection between PRGs and PRFs):

Proposition 3.2 *Suppose $f : \{0,1\}^n \to \{0,1\}^{n+1}$ is a pseudorandom generator that stretches by one bit. Define $f^{(1)}(x) = f(x)$ and for all $i > 0$ $f^{(i+1)}(x) = f(f^{(i)}(x)_{\{1,\dots,n\}})|f^{(i)}(x)_{\{n+1,\dots,n+i\}}$. Let k_n be an integer-valued parameter. Then $f^(k_n)$ is a pseudorandom generator that stretches by k_n bits, and the reduction is linear-preserving.*

The term linear-preserving is a measure used when describing the strength of a reduction from one function ensemble to another. There are three general categories. Linear-preserving essentially defines that the breaking time R_N of the old function linearly maps to the breaking time of the reduction R^1_N, poly-preserving that R_N linearly maps to $R^1_{O(N)}$ and weak-preserving that R^1_N linearly maps to $R^1_{N^{O(1)}}$. So, in the proposition above, we are not losing entropy significantly when extending the random string.

4 OWF to PRG, 1st iteration

Comparing the definitions of OWFs and PRGs, what we are striving for in a OWF to PRG reduction is to make the output of the OWF random in a statistical/computational sense, and make the OWF output more random bits than is input. To satisfy the latter demand, we introduce a proposition (proved by Goldreich & Levin in [4]). We want to generate an (extra) output bit that is computationally indistinguishable from $\beta \in_U \{0,1\}$ (a random bit), and our tool is a OWF. In the following, $\cdot$ denotes the (mod 2) matrix product of two vectors $(\{0,1\}^{n^T} \cdot \{0,1\}^n \to \{0,1\})$:

Proposition 4.1 *Let $f : \{0,1\}^{t_n} \to \{0,1\}^{l_n}$ be a one-way function. Then $X \cdot R$ (where $X, R \in_U \{0,1\}^{t_n}$) is (computationally) hidden with respect to $f(X)|R$. The reduction is linear-preserving with respect to the definition of computationally indistinguishable.*

Now we have a tool for producing a pseudorandom bit by means of an OWF. We continue by examining the special case of a one-way permutation (OWP). We assume uniformly distributed input (a requirement in the definition of a PRG) and being a permutation the function will not lose information ($f(x)$ uniquely determines x) so it will follow that the output of the OWP is uniformly distributed and no change in entropy will occur. This is enough for us to claim that the following construct holds [5]:

Proposition 4.2 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation. Let $x, r \in_U \{0,1\}^n$. Define the P-time function ensemble $g(x) = f(x)|r|x \cdot r$. Then g is a pseudorandom generator. The reduction is linear-preserving with respect to the definition of computationally indistinguishable.*

The proof may be outlined as follows: Let $X, R \in_U \{0,1\}^n$ and $\beta \in_U \{0,1\}$. Because f is a permutation $f(X)|R|\beta \in_U \{0,1\}^{2n+1}$ By proposition 4 $g(X,R)$ and $f(X)|R|\beta$ are computationally indistinguishable and the reduction linear-preserving.

It is noteworthy to see that proposition 4.2 can also be used to reduce a one-to-one OWF into a pseudoentropy generator PEG (explained in the next section). Intuitively this is evident, since the main difference between a PRG and a PEG is that uniformity of the output distribution is not required in PEGs.

5 OWF to PRG, 2nd iteration

The general proof of reducing any OWF (not just a OWP) to a PRG is done in [5] based on the random oracle model, and will produce a weak-preserving reduction from an OWF to a PRG. The general problem statement and some of the partial approaches is described in the following.

In general, there are no range restrictions OWFs, they may lose significant amounts of entropy and the output may be very different from the statistical notion of randomness. Thus, these two 'problems' must be remedied by a successful reduction. Levin & al:s paper [5] use a general hash function to smooth the output of the OWF $f(x)$ to 'randomize' it and also to 'boost' entropy.

The general OWF to PRG reduction will heavily use the notion of entropy. Basic Shannon entropy [1] is a statistical measure on a distribution $H(D) = -log(D(X))$, and it can be applied to a random string / output as a measure of its randomness (although the measure is by no means perfect). Additionally *computational entropy* is defined as follows: Assume that we have a function ensemble $f : \{0,1\}^{t_n} \rightarrow \{0,1\}^{l_n}$ with entropy $H(f(x))$. A second function ensemble $g : \{0,1\}^{m_n} \rightarrow \{0,1\}^{l_n}$ has R-secure computational entropy (pseudoentropy) $s_n <= H(f(x))$ if f and g are computationally indistinguishable.

Based on the different versions of entropy we can define *pseudoentropy generators* (PEG) and *false entropy generators* (FEG) (these will be used as intermediate steps of the OWF-to-PRG reduction). They are similar to PRGs, but where a PRG produces output that is indistinguishable from the uniform distribution, PEGs produce output that has more computational entropy than their input distribution, i.e. a PEG increases the computational entropy. A FEG does even less, its only feature is that the computational entropy on its output is higher than the corresponding statistical entropy.

The proof of the general OWF-PRG reduction will reduce a false entropy generator to a pseudoentropy generator, which in turn is reduced to a pseudorandom generator. The most difficult part of the proof involves the reduction from a general OWF to a false entropy generator. One critical construct in that process is to extract entropy from a OWF.

The notion that a hash function can be used to smooth function output is easy to intuitively understand, but to use it to increase entropy may not be as evident. To make matters precise, we define a *universal* hash function as a two-input function ensemble $h : \{0,1\}^{t_n} \times \{0,1\}^n \rightarrow \{0,1\}^{l_n}$ where for two input-output value pairs $\{x,y\}$ and $\{x^{'},y^{'}\}$ and keeping the second input parameter fixed $\{z \in_U \{0,1\}^n\}$ we have $Pr[h(x,z) = y$ and $h(x^{'},z) = y^{'}] = 1/2^{2l_n}$ To increase the entropy of a OWF with a hash function we need the degeneracy (D_f) of the OWF to be P-time computable (which in itself is a difficult proof). By the degeneracy we define the number of lost randomness (bits) in the output of a OWF, formally $f : \{0,1\}^{t_n} \rightarrow \{0,1\}^{l_n}, z \in range_f, D_f(z) = \lceil log(\#pre_f(z)) \rceil$. If we assume this computability, we can hash the OWF function input into D_f bits of output to maintain entropy. In other words, we are constructing (a poly-preserving) reduction from one OWF to another: $f(x,y) = f(x)|h_y(x)_{\{1,\dots,D_f\}}|y$, where y is second input to the function ensemble h with two inputs.

6 PRG to OWF

Reducing a PRG to an OWF is straight-forward, the PRG actually has to be an OWF in order to satisfy its definition, i.e.

Proposition 6.1 *Let $f : \{0,1\}^{t_n} \rightarrow \{0,1\}^{l_n}$ be a PRG. Then $g(x) = f(x)$ is a OWF.*

PROOF OUTLINE: We assume that there is an R_{t_n}-breaking adversary (A_1) for g. Based on adversary A_1 we may build an adversary A_2 in the following manner. Respecting the polynomial time limit of A_2 we distinguish between the output of the original PRG from uniformly random output f by using A_1 to invert the given output. If A_1 returns an error or doesn't give a result in polynomial time we conclude that the output was random, otherwise we assume $f(x)$. But this is in contradiction with $f(x)$ being a PRG, implying that $g(x)$ is a OWF.

7 PRF

A *pseudo-random function* is intuitively something like a keyed hash function or a symmetric encryption function - we input two values (one is the key) and as output we get something that is indistinguishable from the input. This property is defined by the adversary mechanism in the following:

Definition 7.1 Let $f : \{0,1\}^{k_n} \times \{0,1\}^{t_n} \rightarrow \{0,1\}^{l_n}$ be a P-time computable function ensemble with two inputs (a function family). The success probability of an adversary A for distinguishing a generic element in f_s (from a function F drawn at random) is $sp_k(A) = Pr[A^{f_s}(1^k) = 1]$, where $s \in_U \{0,1\}^{k_n}$. The function ensemble f is a R-secure pseudo-random function if there is no R-breaking adversary for f, distinguishing f from a randomly drawn function (with $sp_k(A) = Pr[A^F(1^k) = 1]$ where $F \in_U \{0,1\}^{t_n} \times \{0,1\}^{l_n}$

Said another way, there should be no adversary that upon input of the key length k can tell the difference between input-output pairs of the function f and input-output pairs drawn from two uniform random distributions, the latter pairing having the additional 'oracle' feature that if queried twice with the same value it also returns the same result.

8 PRF to PRG

The reduction of a PRF to a PRG can be done with a simple 'oracle' construct [7]. A given PRG will (on a fixed input taken from the uniform distribution) output a pseudorandom string of some polynomial length, say r^c. Assuming this we may construct an oracle (a machine) that returns the value of the bit at position $p < r^c$. To be able to describe all positions we need $p \in 0, 1^{c \log r}$. On the other hand, using the same notations, we can define a PRF function ensemble $f : \{0,1\}^{k_n} \times \{0,1\}^{c \log r} \rightarrow \{0,1\}^1$. From this statement (assuming the PRF) it will follow that if we cannot distinguish f from a randomly drawn function we cannot distinguish the output bit of f from the corresponding uniform distribution (if the latter wouldn't hold we could build an adversary to break the PRF). We must additionally require that $k_n < c \log r$, to satisfy the requirement that PRG must increase the length of its input.

9 PRG to PRF

The proof for reducing a PRG to a PRF presented here is due to Goldreich, Goldwasser and Micali [2], known as the GGM construction. It is also well explained in [7].

Before going into the proof itself, we introduce the *hybrid argument*, which is mentally close to mathematical induction. Assume that we want to prove that a distribution A is computationally indistinguishable from distribution B. Rather than proving this directly (assume that it is difficult), we may define intermediate distributions $A = A_0, A_1, \ldots, A_n = B$ and prove the computational indistinguishability for each pair A_x, A_{x+1} separately. It can be algebraically proven [7] that this implies computational indistinguishability between A and B as well (although we may somewhat increase the success probability of the adverasy in the process). The essence of the argument is that the adversary evaluating our function output doesn't mind if the output is not a 'real algorithm' or a 'real random output', we can bridge the gap with any construct we want as long as the adversary is 'fooled'.

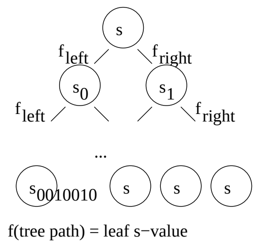

Figure 2: *Tree construct*

We start the construct by taking a PRG that doubles its input, e.g. $f : \{0,1\}^n \rightarrow \{0,1\}^{2n}$, effectively splitting the range in two and defining two new functions: $f_{left}(x) = f(x)_{\{1,n\}}, f_{right}(x) = f(x)_{\{n+1,2n\}}$. By recursively applying the split functions on the results of the previous round we build a balanced binary tree of depth n where $s_0 = f_{left}(x)$ and $s_1 = f_{right}(x)$ define the first level, and e.g. s_{00} and s_{01} are the children of s_0 and calulated by applying the value of s_0 as input to the functions f_{left} and f_{right}. We now claim that the tree represents a family of PRF functions where the two input parameters are the seed value for the tree (the key) and the position (index) of one leaf in the tree. The output is the corresponding s_x - value.

To prove this theorem, the hybrid argument can be applied in two phases. The final target is to prove computational indistinguishability between the assumed PRF defined above (T_0) and 'an empty tree' with all leaves filled with values drawn from a uniform random distribution (T_0). We first apply the hybrid argument to oracles $T_0 \dots T_n$ where $T_r, 0 \leq r \leq n$ denotes a tree filled in such a way that the tree level x counted from the root is filled with random values, and all nodes below are calculated in the manner described above. At this point, the proof of computational indistinguishability has by the hybrid argument been reduced to proving indistinguishability between T_r and T_{r+1}.

The second application of the hybrid argument (where we want to add intermediate steps between T_r and T_{r+1}) poses a slight problem, as the growth of the binary tree is exponential with respect to the level, and according to the definition of the PRF the computational indistinguishability must be done in polynomial time. So we define the number t to be the number of queries that the adversary is allowed to make, and define the intermediate oracle $T_{rk}, 0 \leq k \leq t$ in such a way that it answers according to T_r to the first t queries, and according to T_{r+1} to the $t+1$:th query. Additionally we require the oracle T_{rk} to be consistent with respect to previous queries (if the tree nodes have already been filled in we use the existing values).

Now, a successful adversary would have to distinguish between an oracle filling its node with a random value and its children nodes with s_0 and s_1 according to $f(x)$ and an oracle filling its children nodes with random values. If this would be the case, the oracle could distinguish between the output of a PRG and random output which would contradict the pseudorandomness of the PRG. By the hybrid argument the theorem holds.

10 Conclusions

Most of the connections between the OWF, PRG and PRF function classes that have been shown here are not computationally efficient, and they merely serve the purpose of providing infrastructure for the general theory that is built up around these building blocks. For some of the reductions there are alternatives, variants and optimizations, and there are also variations in the actual proof methodology. Related discussions have also put the random oracle model under scrutiny [6], although any implications on these constructs are not given.

References

[1] C. Shannon A Mathematical Theory of Communication Bell Systems Technical Journal, 27, 1948, pp. 379-423 and pp. 623-656

[2] O. Goldreich, S. Goldwasser and S. Micali How to Construct Random Functions Journal of the Association for Computing Machinery, Vol 33, Mo 4, Oct 86, pp. 792-807

[3] Boppana, R., Hirschfeld, R. Pseudo-random generators and complexity classes Advances in Comp. Research, S. Micali, ed., JAI Press, 5 (1989), pp. 1–26.

[4] O. Goldreich, L. Levin. A hard-core predicate for any one-way function. Proceedings of the 21st ACM Symposium on Theory of Computing, 1989.

[5] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random number generation from one-way functions. Proc. of the 21st Annu. ACM Symp. on the Theory of Computing, pages 12-24, 1989.

[6] R. Chang, B. Chor, O. Goldreich, J. Hartmanis, J. Hastad, D. Ranjan, and P. Rohatgi. The Random Oracle Hypothesis is False. J. Comput. Syst. Sci. 49:1 (1994), 24-39.

[7] Y. Dodis & al. Lecture Notes, Introduction to Cryptography, NY Univ. (G22.3033-003) www.cs.nyu.edu/courses