Created: 27 Jan 99.
Copyright ©1999 by Phillip Rogaway

ECS 227 — Modern Cryptography — Winter 99
Lecture 3

From Simple Primitives to Complex Ones: The PRG → PRF → PRP Constructions (Part II)

21 January 1999

1 Review

In the last lecture:

- We (again) defined what is a pseudorandom generator (PRG) and how to measure its security. We showed how to construct a pseudorandom generator $g_* : \{0,1\}^\alpha \to \{0,1\}^{2\alpha}$ given a pseudorandom generator $g : \{0,1\}^\alpha \to \{0,1\}^{\alpha+1}$. We then analyzed that construction.
- We defined what is a pseudorandom function (PRF) (often called "a family of pseudorandom functions" or "pseudorandom function family") and we described a construction, due to [GoGoMi86], for constructing a PRF from a length-doubling PRG.

Before proceeding you should review those notes and make sure you know the definitions.

Today we are going to do two things:

(1) We'll prove the prove correctness of the PRG→PRF construction we described last time.

(2) We'll describe what is a pseudorandom permutation (PRP) (often called "a family of pseudorandom permutations" or "pseudorandom permutation family") and we'll show how to construct a PRP from a PRF. The construction is the same as that underlying the DES construction; we'll use a 3-round Feistel network.

Let us begin.

2 From PRG to PRF

Let me remind you of the construction. We have a PRG g which goes from α bits to 2α bits. Let's say we want to construct a PRF $F : \{0,1\}^\alpha \times \{0,1\}^\alpha \to \{0,1\}^\alpha$. To compute $F_a(x)$, do the following. Make a binary tree and put a at the root. Label this node $a_\varepsilon = a$. Now let a_s be some arbitrary node in the tree, where $|s| < \alpha$. Then the left child for a_s is

$$a_{s0} = g(x)[1..\alpha]$$

while the right child for a_s is

$$a_{s1} = g(x)[\alpha + 1..2\alpha] \; .$$

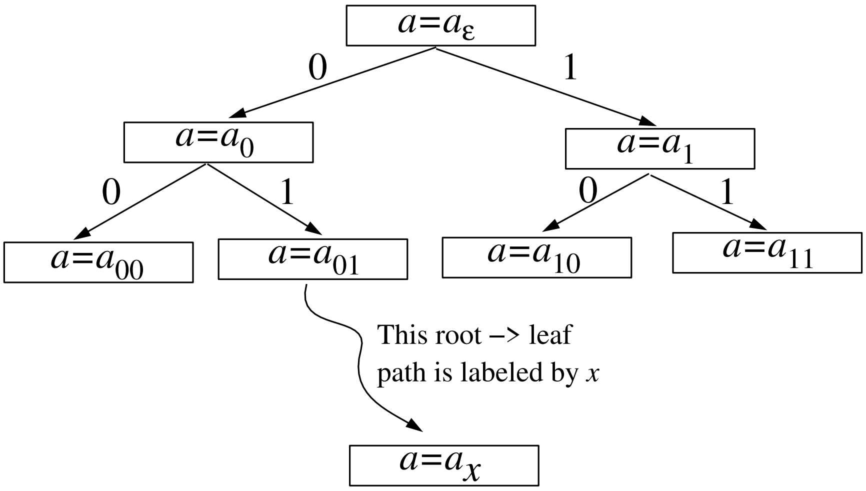

Figure 1: Making a PRF from a PRG. The left and right children of a node a_s are the first α bits of $g(a_s)$ and the remaining α bits of $g(a_s)$. Denote these a_{s0} and a_{s1}, respectively. Then Define $F_a(x) = a_x$.

See Figure 1.

The way cryptography is often done we have informal theorems before we have proofs, and we normally have proofs before theorem statements. So in class I stated the following theorem but I didn't fill in the value for t' and δ'. We filled those in later, after we did the proof. In the less dynamic format of lecture notes I'll do it all from the start.

Theorem 2.1 Let $g : \{0,1\}^\alpha \to \{0,1\}^{2\alpha}$ be a PRG and define F from g according to the construction we have described. Let D_F be an adversary (for distinguishing F from a random function) and suppose that D_F runs in time t, asks q oracle calls, and achieves advantage $\delta = \mathrm{Adv}_F^{\mathrm{prf}}(D_F)$. Then there exists an adversary D_g (for attacking the PRG g) that runs in time at most t' and achieves advantage at least δ' where

$$\begin{aligned} t' &= t + c\alpha q \cdot \mathrm{Time}_g, \text{ and} \\ \delta' &= \frac{\delta}{\alpha q}, \end{aligned}$$

where c is an absolute constant and Time_g denotes the time to compute $g(x)$.

Proof. This will be another hybrid argument.

Suppose we would like to exactly simulate an $F_a(\cdot)$-oracle (for a random a) for the benefit adversary D_F. This is easy to do. Here is one way to describe it.

Algorithm $D_g^0(y)$
Make a binary tree $T = (\{V_\varepsilon\}, E)$ with one node—the root, V_ε, labeled by $a_\varepsilon \leftarrow \{0,1\}^\alpha$.
Run the adversary D_F. When D_F makes an oracle query, x:
  Repeat until there is a node V_x in T:
    Let u be the longest prefix of x such that T has a node V_u.
    Create left and right children for V_u, calling them V_{u0} and V_{u1}, respectively.
    Assign V_{u0} label $a_{u0} = g(a_u)[1..\alpha]$, and assign V_{u1} label $a_{u1} = g(a_u)[\alpha+1..2\alpha]$.
  Return in response to query x the string a_x which labels V_x.
When D_F finally outputs a bit, b, output the same bit.

Here I have indicated that D_g^0 takes an argument, y, and this argument is 2α-bit string. But, in fact, $D_g^0(y)$ does not depend on y.

Here is a different "extreme" for simulating an oracle for D_F. This time D_F is provided what is going to look just like a random function.

> **Algorithm** $D_g^{\alpha q}(y)$
> Make a binary tree $T = (\{V_\varepsilon\}, E)$ with one node—the root, V_ε, labeled by $a_\varepsilon \leftarrow \{0,1\}^\alpha$.
> Run the adversary D_F. When D_F makes an oracle query, x:
>   Repeat until there is a node V_x in T:
>   Let u be the longest prefix of x such that T has a node V_u.
>   Create left and right children for V_u, calling them V_{u0} and V_{u1}, respectively.
>   Assign V_{u0} label $a_{u0} \leftarrow \{0,1\}^\alpha$ and assign V_{u1} label $a_{u1} \leftarrow \{0,1\}^\alpha$.
>   Return in response to query x the string a_x which labels V_x.
> When D_F finally outputs a bit, b, output the same bit.

Here I have again indicated that $D_g^{\alpha q}$ takes an argument, y, and this argument is again 2α bits. But, as before, $D_g^{\alpha q}(y)$ does not depend on y.

Now clearly

$$\Pr[a \leftarrow \{0,1\}^\alpha : D_F^{F_a(\cdot)} = 1] \quad = \quad \Pr[D_g^0(y) = 1]$$

while

$$\Pr[\rho \leftarrow \mathrm{Rand}_{\alpha\to\alpha} : D_F^{\rho(\cdot)} = 1] \quad = \quad \Pr[D_g^{\alpha q}(y) = 1]$$

We know that

$$D_F^{F_a(\cdot)} = 1] - D_F^{\rho(\cdot)} = 1] = \delta$$

and so, in particular,

$$\Pr[D_g^0(y) = 1] - \Pr[D_g^{\alpha q}(y) = 1] = \delta\ .$$

We will now create some "intermediate" adversaries. Intermediate adversary $D_g^i(y)$ will randomly generated i pairs of labels and use them for the first i times that it needs labels. It will use $y = y_L y_R$ for the i-th label. And it will pseudorandomly generate the remaining pairs of labels. The code is as follows:

> **Algorithm** $D_g^i(y_L \parallel y_R)$
> Make a binary tree $T = (\{V_\varepsilon\}, E)$ with one node—the root, V_ε, labeled by $a_\varepsilon \leftarrow \{0,1\}^\alpha$.
> Let $j \leftarrow 0$.
> Run the adversary D_F. When D_F makes an oracle query, x:
>   Repeat until there is a node V_x in T:
>   Let u be the longest prefix of x such that T has a node V_u.
>   Create left and right children for V_u, calling them V_{u0} and V_{u1}, respectively.
> $j \leftarrow j + 1$.
>   Assign V_{u0} label a_{u0} and assign V_{uq} label a_{u1} where:
>   If $j \le i$ then $a_{u0} \leftarrow \{0,1\}^\alpha$ and $a_{u1} \leftarrow \{0,1\}^\alpha$.
>   If $j = i$ then $a_{u0} \leftarrow y_L$ and $a_{u1} \leftarrow y_R$.
>   If $j > i$ then $a_{u0} \leftarrow g(a_u)[1..\alpha]$ and and $a_{u1} \leftarrow g(a_u)[\alpha + 1..2\alpha]$.
>   Return in response to query x the string a_x which labels V_x.
> When D_F finally outputs a bit, b, output the same bit.

You should check that when y is pseudorandom the experiment described above coincides, when $i = 0$ and $i = q\alpha$, with the ones we described earlier.

Define p_i as the probability that $D_g^i(y)$ outputs 1 when y is selected pseudorandomly,

$$p_i \quad = \quad \Pr[x \leftarrow \{0,1\}^\alpha \,; y \leftarrow g(x) : D_g^i(y) = 1] \, .$$

We know at this point that $p_0 - p_{\alpha q} \geq \delta$. As a consequence, there exists an i such that $p_i - i_{i+1} \geq \delta/(q\alpha)$. Fix such an i and proceed.

Now we make the key observation that when y is selected randomly, the probability that $D_g^i(y)$ outputs 1 is exactly p_{i+1}:

$$p_{i_1} \quad = \quad \Pr[y \leftarrow \{0,1\}^{2\alpha} : D_g^i(y) = 1]$$

This is easy to see: we used y, which was random, for the i-th pair of random values, and so the first pseudorandom value we used was the $(i+1)$-st. Thus we now constructed a distinguisher, D_g^i, which achieves advantage $\delta/(\alpha q)$ in distinguishing random and pseudorandom strings of length 2α.

If you don't like the non uniformity in our construction—the fact that we don't know what i to use—just choose an i at random. The "average gap" between p_i and p_{i+1} has to be at least $\delta/(\alpha q)$, so this is fine.

We have still to check that the running time of $D_g = D_g^i$ is as claimed. The main thing in the running of D_g is the running of D_F. Additional overhead consists of dynamically maintaining the tree T and, in particular, populating its nodes by choosing random values or applying g a total of at most αq times. Assuming that g requires at least linear time, $O(\alpha q)$ times the time to compute g clearly suffices. ▌

3 From PRF to PRP

A PRP is a function $F : \{0,1\}^\alpha \times \{0,1\}^\beta \rightarrow \{0,1\}^\beta$ such that $F_a(\cdot)$ is always a permutation. As we indicated, we are going to use the three-round Feistel construction to make a good PRP from a good PRF. In Figure 2 is a picture to remind you how this works. We'll let the left hand half and right hand half of each message block $M = M_L M_R$ have n-bits. Thus we'll produce a PRP on $2n$ bits given our PRF on n bits.

We will concentrate on the information-theoretic version of this construction, which already has all the main ideas. We will take three *random* functions and apply the three round Feistel construction, and we will see how "close" this gets us to a random permutation. Being close to a random permutation in the information theoretic sense means that if we don't ask too many questions then our advantage in distinguishing the construction from a truly random permutation is small. For convenient, we will actually measure the distance to a random function, not permutation, but this is easily corrected. Here is the theorem.

Theorem 3.1 Let D be an adversary that asks q oracle queries. Then

$$\begin{aligned} \mathrm{Adv}^{\mathrm{prf}}_{\mathrm{Fe}}(D) &= \Pr[\rho_1, \rho_2, \rho_3 \leftarrow \mathrm{Rand}_{n \to n} : D^{\mathrm{Fe}[\rho_1, \rho_2, \rho_3](\cdot)} = 1] - \Pr[\rho \mathrm{Rand}_{2n \to 2n} : D^{\mathrm{Fe}(\rho(\cdot))} = 1] \\ &\leq \frac{q^2}{2^n} \end{aligned}$$

The proof is quite clever, and introduces a new proof technique.

Proof. Let us call "Game Fn" the game in which we choose three random functions $\rho_1, \rho_2, \rho_3 \leftarrow \mathrm{Rand}_{n \to n}$ and then answer D's queries according to $\mathrm{Fe}[\rho_1, \rho_2, \rho_3](\cdot)$. Let us call "Game Rn" the

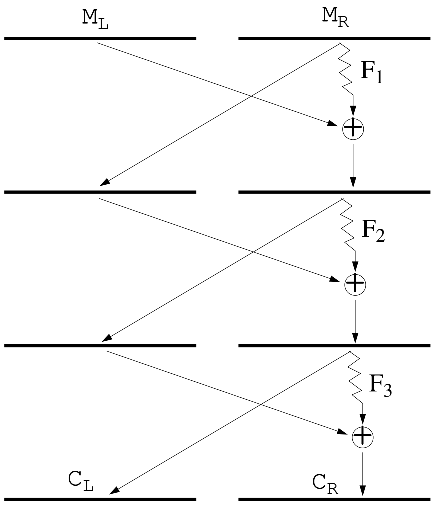

Figure 2: Making a PRP from a PRF: the three-round Feistel construction. Here $M = M_L M_R$ is mapped to $C = C_L C_R$ using using PRFs F_1, F_2, F_3.

game in which we choose a random function $\rho \in \text{Rand}_{2n \to 2n}$ and then answer D's queries according to $\rho(\cdot)$. Let's denote by P_{Fe} the probability the D outputs 1 in Game Fe, and call by P_{Rn} the probability that D outputs 1 in Game Rn. We are trying to show that $P_{\text{Fe}} - P_{\text{Rn}} \le q^2 2^{-n}$.

Without loss of generality, assume that D never repeats a query. Adversary D has no need to repeat queries; she might as well remember the answer she already obtained.

We begin by describing a different way to play Game Fe. We call this game "Game B'." Game B' is going to look the same to adversary D as Game Fe. This will be clear. But Game B' will be played completely differently. Instead of choosing three random functions ρ_1, ρ_2, ρ_3, we'll only choose some random strings $x_1, \dots, x_q$, $y_1, \dots, y_q$, and $z_1, \dots, z_q$. Each of these strings is of length n. The *only* random choices we will make in playing game B' is in the choice of x, y, and z. We describe Game B' in Figure 3. It is played as follows. Begin by choosing random strings $x_1, \dots, x_q, y_1, \dots, y_1, z_1, \dots, z_q$. Now answer the i-th query with $\beta_i \parallel \gamma_i$, where β_i and γ_i are described in the figure.

An examination of the game will make it obvious that this is the same, as far as the adversary can see, to Game Fe. Thus $P_{\text{Fe}} = \Pr[D^{B'} = 1]$.

We now modify Game B' to a game B which is identical, from the adversary's point of view, to B (and therefore to Fe). This modification is unusual: we will throw R_{v_i} into the second XOR, and we will throw in α_{w_i} into the final XOR. The new game is show in Figure 4.

It is not completely obvious that these additional XORs do not change the game. The trick is to look at $(R_{v_i} \oplus y_i, \alpha_{w_i} \oplus z_{w_i})$ in Game B and notice that these are uniformly distributed random variables, independent of everything else, just as (y_{v_i}, z_{w_i}) was in game B'. For further details, look at Luby's writeup.

We now have that $P_{\text{Fe}} = \Pr[D^B = 1]$. The probability is taken over random q-vectors of n-bit strings x, y, z.

We consider one final game, Game C. This game is identical to B except that we output $y_i \parallel z_i$ (instead of $\beta_i \parallel \gamma_i$). Obviously $P_{\text{Rn}} = \Pr[D^C = 1]$. Again the probability is over random x, y, z.

We will now make some observations and calculations about Games B and C which will allow us to conclude with the theorem. The idea is that Games B and C usually coincide. We will manage to bound adversarial advantage by looking at the chance that games B and C do not coincide.

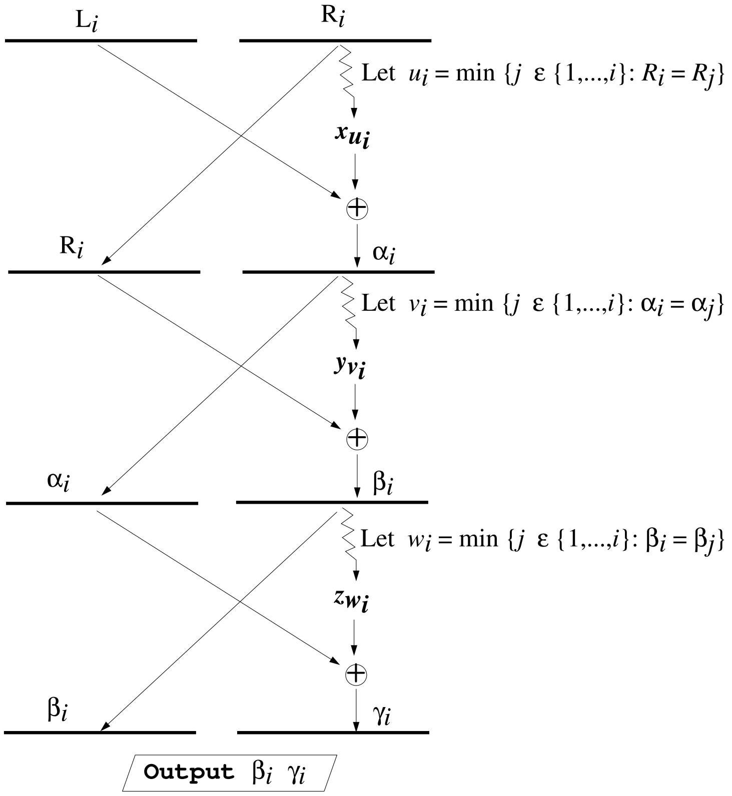

Figure 3: Game B'. This game is identical, as far as the adversary can tell, to Game Fe. Begin by choosing $x_1, \ldots, x_q, y_1, \ldots, y_q, z_1, \ldots, z_q$ at random. Then answer the i-th query, L_iR_i, by $\beta_i\gamma_i$, computed as in the figure.

Let us now define some events. These events are defined in Game C. (It is important that we do this in Game C, not Game B.) Define the event REPEAT_α as true if $\alpha_i = \alpha_j$ for some $i < j$—that is, some α_i arises twice. Define the event REPEAT_β as true if $\beta_i = \beta_j$ for some $i < j$—that is, some β_i arises twice. Define the event REPEAT as the disjunct of α_i and β_i—that is, either an α_i repeats or a β_i repeats. Again, these events are defined in Game C.

Claim 3.2 $\Pr[\mathsf{REPEAT}_\alpha] \leq \dfrac{1}{2}\dfrac{q^2}{2^n}$.

Look at query i. If R_i itself is a repetition of an earlier R_j, then we know for sure that $\alpha_i \neq \alpha_j$, since all queries are assumed to be distinct. It is possible, however, that α_i could coincide with some α_j where R_j was different from R_i. But we have provided the adversary no information about internal x_i and α_i values, and so the chance that any particular one of these α_j values coincides with α_i is 2^{-n}. If R_i is a new value, then x_{u_i} will be a random n-bit string and so the chance that α_i will collide with any particular prior α_j is again 2^{-n}. Thus the chance that α_i will collide with an earlier query is at most $(i-1)2^{-n}$, and the chance that there will eventually be a collision in α_i-values is at most $\sum_{i=1}^{q} \frac{i-1}{2^n} \leq \frac{1}{2} \cdot \frac{q^2}{2^n}$. $\diamondsuit$

Claim 3.3 $\Pr[\mathsf{REPEAT}_\beta | \overline{\mathsf{REPEAT}_\alpha}] \leq \dfrac{1}{2}\dfrac{q^2}{2^n}$.

This one is easy to see. By assumption, the v_i values are all distinct, so y is being evaluated on

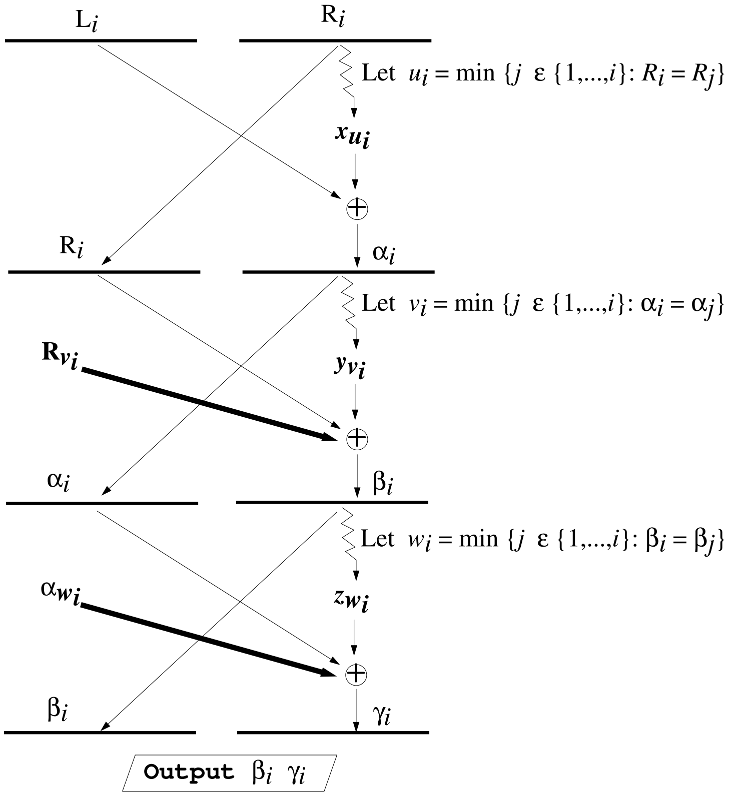

Figure 4: Game B. We modify B' by adding the emboldened XORs. This game is once again identical, from the adversary's perspective, to Game Fe.

distinct points. The chance that two y_i values coincide is thus given by the birthday upper bound, once again $\sum_{i=1}^{q} \frac{i-1}{2^n} \leq \frac{1}{2} \cdot \frac{q^2}{2^n}$. $\diamondsuit$

Putting this together we have that

Claim 3.4 $\Pr[\mathsf{REPEAT}] \leq \dfrac{q^2}{2^n}$.

The reason is that

$$\begin{aligned}\Pr[\mathsf{REPEAT}] &= \Pr[\mathsf{REPEAT}_\alpha] + \Pr[\mathsf{REPEAT}_\beta \wedge \overline{\mathsf{REPEAT}_\alpha}] \\ &\leq \Pr[\mathsf{REPEAT}_\alpha] + \Pr[\mathsf{REPEAT}_\beta | \overline{\mathsf{REPEAT}_\alpha}] \cdot \Pr[\overline{\mathsf{REPEAT}_\alpha}] \\ &\leq \Pr[\mathsf{REPEAT}_\alpha] + \Pr[\mathsf{REPEAT}_\beta | \overline{\mathsf{REPEAT}_\alpha},\end{aligned}$$

and we have just bounded each of the above addends by $q^2/2^{n+1}$. $\diamondsuit$

Now for the magical observation:

Claim 3.5 $\Pr[D^B = 1 | \overline{\mathsf{REPEAT}}] = \Pr[D^C = 1 | \overline{\mathsf{REPEAT}}]$.

Both probabilities are over random choices of x, y, z. Now on the right-hand we output y_i, z_i in response to the ith query. On the left-hand side, assuming that REPEAT does not hold *in Game C*, once again we output y_i, z_i. Huh? This would be clear if we had said "assuming that REPEAT does not hold *in Game B*," and we defined this even in Game B in the obvious manner. But the thing to notice is that as long as REPEAT does not hold in Game C, Game C and Game B behave identically, always returning $y_i z_i$ in response to query i. This is easily established by induction. $\diamondsuit$

We have just one more detail to complete the proof. We claim that, because of the last claim,

$$\begin{aligned} P_{\mathrm{Fe}} - P_{\mathrm{Rn}} &= \Pr[D^B = 1] - \Pr[D^C] = 1 \\ &\leq \Pr[\mathsf{REPEAT}] \end{aligned}$$

What justifies that? Very simple. Let A, B, C be arbitrary events events and assume $\Pr[A|\overline{C} = \Pr[B|\overline{C}]$. Now

$$\Pr[A] - \Pr[B] = \Pr[A|\overline{C}]\Pr[\overline{C}] + \Pr[A|C]\Pr[C] - \Pr[B|\overline{C}]\Pr[\overline{C}] - \Pr[B|C]\Pr[C]$$

and so $\Pr[A|\overline{C}] = \Pr[B|\overline{C}]$ tells us that first and third addends cancel. Now upperbound the second added by dropping the $\Pr[A|C]$ (that is, upperbound this by 1) and drop the final added (which is negative) entirely, thereby getting an upperbound of $\Pr[C]$, as desired. ▮

References

PRFs from PRGs. This is due to:

[GGM] "How to Construct Random Functions," Oded Goldreich, Shafi Goldwasser and Silvio Micali, *Journal of the ACM*, Vol. 33, No. 4, 1986, pp. 792–807.

(I made up the proof in preparing the lecture, so I don't know that its the same same as the one in the paper.)

PRPs from PRFs. This is due to:

[LR] Michael Luby and Charles Rackoff, "How to Construct Pseudorandom Permutations from Pseudorandom Functions." *SIAM J. Comput*, Vol. 17, No. 2, 1988, pp. 373–386.

The proof I gave is taken from Lecture 13 of

[L] Michael Luby, *Pseudorandomness and Cryptographic Applications*, Princeton University Press, 1996.