Лекция 4

Псевдослучайные функции

(Конспект: Г. Ярославцев)

Внимание! Этот текст не является конспектом лекции, а является переводом соответствующих мест из книги O. Goldreich "Foundation of cryptography". (Д.М. Ицыксон)

4.1 Дополнения к предыдущей лекции

Определение 4.1 (Неразличимость с полиномиальным количеством сэмплов). Две последовательности случайных величин $X = \{X_n\}_{n\in\mathbb{N}}$ и $Y = \{Y_n\}_{n\in\mathbb{N}}$ наызваются **неразличимыми при полиномиальном количестве сэмплов**, если для любого вероятностного полиномиального алгоритма D, для любых двух положительных полиномов $m(\cdot)$ и $p(\cdot)$ и любых достаточно больших n

$$|\Pr[D(X_n^{(1)},\ldots,X_n^{(m(n))})=1]-\Pr[D(Y_n^{(1)},\ldots,Y_n^{(m(n))})=1]|<\frac{1}{p(n)}$$

где величины от $X_n^{(1)}$ до $X_n^{(m(n))}$ и от $Y_n^{(1)}$ до $Y_n^{(m(n))}$ случайны и независимы и все $X_n^{(i)}$ совпадают с X_n, а все $Y_n^{(i)}$ совпадают с Y_n.

Определение 4.2 (Полиномиально моделируемые распределения). Последовательность случайных величин $X = \{X_n\}_{n\in\mathbb{N}}$ называется полиномиально моделируемой, если существует вероятностный полиномиальный алгоритм S, такой что для любого n случайные величины $S(1^n)$ и X_n распределены одинаково.

Теорема 4.1. *Пусть $X = \{X_n\}_{n\in\mathbb{N}}$ и $Y = \{Y_n\}_{n\in\mathbb{N}}$ это полиномиально моделируемые последовательности случайных величин и пусть X и Y вычислительно неразличимы. Тогда X и Y неразличимы и с полиномиальным количеством сэмплов.*

Доказательство. Доказательство будет основано на сведении. Мы покажем, что из существования алгоритма, который отличает X и Y с использованием полиномиального количества сэмплов следует существование алгоритма, который отличает X от Y с использованием одного сэмпла. Доказательство использует гибридный метод.

Предположим противное, то есть что существует вероятностный полиномиальный алгоритм D и полиномы $m(\cdot)$ и $p(\cdot)$, такие что для бесконечно большого количества n выполняется

$$\Delta(n) \stackrel{def}{=} |\Pr[D(X_n^{(1)}, \ldots, X_n^{(m(n))}) = 1] - \Pr[D(Y_n^{(1)}, \ldots, Y_n^{(m(n))}) = 1]| > \frac{1}{p(n)}$$

где $X_n^{(i)}$ и $Y_n^{(i)}$ такие же, как в определении неразличимости с полиномиальным количеством сэмплов. Тогда мы получим противоречие, построив вероятностный полиномиальный алгоритм D', который будет различать случайные величины X и Y.

Для всех k, где $0 \leq k \leq m$, определим гибридную случайную величину H_n^k как последовательность длины k, состоящую из k независимых копий X_n, за которыми следуют $m-k$ независимых копий Y_n. А имеенно

$$H_n^k \stackrel{def}{=} (X_n^{(1)}, \ldots, X_n^{(k)}, Y_n^{(k+1)}, \ldots, Y_n^{(m)})$$

где величины от $X_n^{(1)}$ до $X_n^{(k)}$ и от $Y_n^{(k+1)}$ до $Y_n^{(m)}$ независимы и случайны, так что все $X_n^{(i)}$ совпадают с X_n, а все $Y_n^{(i)}$ совпадают с Y_n. Таким образом, $H_n^m = (X_n^{(1)}, \ldots, X_n^{(m)})$, а $H_n^0 = (Y_n^{(1)}, \ldots, Y_n^{(m)})$.

Построим на основе алгоритма D алгоритм D'. На входе α (находящемся в образе X_n или Y_n) алгоритм D' выбирает k равномерно их множества $\{0, 1, \ldots, m-1\}$. Используя полиномиальный алгоритм сэмплирования величины X, алгоритм D' генерирует k независимых сэмплов X_n. Эти сэмплы мы обозначим за $x^1, \ldots, x^k$. Аналогично, используя эффективный алгоритм сэмплирования величины Y, алгоритм D' генерирует $m-k-1$ независимых сэмплов Y_n, которые мы обозначим за $y^{k+2}, \ldots, y^m$. Наконец, алгоритм D' запускает алгоритм D и останавливается, выдавая $D(x^1, \ldots, x^k, \alpha, y^{k+2}, \ldots, y^m)$. Очевидно, что алгоритм D' может быть реализован вероятностно за полиномиальное время. Также легко проверить следующие утверждения:

Утверждение 4.1.

$$\Pr[D'(X_n) = 1] = \frac{1}{m} \sum_{k=0}^{m-1} \Pr[D(H_n^{k+1}) = 1]$$

и

$$\Pr[D'(Y_n) = 1] = \frac{1}{m} \sum_{k=0}^{m-1} \Pr[D(H_n^k) = 1]$$

Доказательство. По построению алгоритма D' мы имеем

$$D'(\alpha) = D(X_n^{(1)}, \ldots, X_n^{(k)}, \alpha, Y_n^{(k+2)}, \ldots, Y_n^{(m)})$$

где k равномерно распределено на множестве $\{0, 1, \ldots, m-1\}$. Используя определение гибридов H_n^k, получаем, что утверждение верно. □

Утверждение 4.2. *Для $\Delta(n)$ (в том виде, в котором оно определялось выше) справедливо*

$$|\Pr[D'(X_n) = 1] - \Pr[D'(Y_n) = 1]| = \frac{\Delta(n)}{m(n)}$$

Доказательство. Используя предыдущее утверждение в первом равенстве получаем:

$$|\Pr[D'(X_n)=1]-\Pr[D'(Y_n)=1]| = \frac{1}{m}\left|\sum_{k=0}^{m-1}\Pr[D(H_n^{k+1})=1]-\Pr[D(H_n^k)=1]\right|$$
$$= \frac{1}{m}\left|\Pr[D(H_n^m)=1]-\Pr[D(H_n^0)=1]\right| = \frac{\Delta(n)}{m} \quad (4.1)$$

где последнее равенство следует из того, что $H_n^m = (X_n^{(1)},\ldots,X_n^{(m)})$ и $H_n^0 = (Y_n^{(1)},\ldots,Y_n^{(m)})$ и определения $\Delta(n)$. □

Таким образом, из нашего предположения о том, что $\Delta(n) > \frac{1}{p(n)}$ для бесконечно большого числа n следует, что вероятностный полиномиальный алгоритм D различает X и Y, что противоречит предположению теоремы. Таким образом, теорема доказана. □

4.2 Введение

В этой лекции будут рассмотрены определения и конструкции псевдослучайных функций (использующие псевдослучайный генератор в качестве составной части).

Мотивация. Вспомним о том, что псевдослучайные генераторы позволяют нам генерировать большое количество псевдослучайных битов, используя при этом меньшее количество случайных битов. Если говорить более точно, то можно сгенерировать $poly(n)$ псевдослучайных битов, использовав для этого n случайных битов. Поскольку любой эффективный алгоритм использует лишь полиномиальное количество случайных значений, наличие доступа к полиномиальному количеству псевдослучайных элементов может показаться достаточным для любого такого алгоритма. Однако, такое заключение слишком поспешно, поскольку оно неявно предполагает, что эти элементы (то есть адреса, к которым будет осуществляться доступ) будут фиксированы заранее. В некоторых естественных приложениях может потребоваться доступ к адресам, которые определяются "динамически" в ходе работы приложения. Например, нам может понадобиться присвоить значения полиномиальному от n количеству n-битовых строк. Проблема, которая рассматривается в этой лекции, состоит в том, чтобы решить эту задачу, сгенерировав всего лишь n случайных битов и *потратив место на хранение всего лишь n битов*. Ключом к решению является понятие псевдослучайных функций. Если говорить неформально, то псевдослучайная функция, используемая группой пользователей, дает им функцию, которая кажется случайной противникам за пределами этой группы.

4.3 Определения

Грубо говоря, псевдослучайные функции это функции которые не могут быть отличены от истинно случайных функций эффективной процедурой, которая может получать

значения функций на произвольных аргументах по своему выбору. Таким образом, отличающая процедура может запрашивать значения рассматриваемых функций в различных точках, которые возможно зависят от предыдущих полученных ответов, и все равно не может определить, были ли ответы предоставлены функцией, взятой из псевдослучайного ансамбля функций или же выбранной из равномерного ансамбля функций. Для простоты мы будем рассматривать ансамбли сохраняющих длину функций, а также для простоты читатель может в последующем изложении считать, что $l(n) = n$.

Определение 4.3 (Ансамбль функций). Пусть $l\colon \mathbb{N} \to \mathbb{N}$ (например, $l(n) = n$). **Ансамблем l-битовых функций** называется последовательность $F = \{F_n\}_{n\in\mathbb{N}}$ случайных величин, такая что случайная величина F_n принимает значения в множестве функций, переводящих l-битные строки в l-битные строки. **Равномерный ансамбль l-битовых функций**, обозначаемый как $H = \{H_n\}_{n\in\mathbb{N}}$, имеет в качестве H_n равномерное распределение на множестве всех функций, переводящих l-битные строки в l-битные строки.

Чтобы формализовать понятие псевдослучайных функций мы используем (вероятностные полиномиальные) оракульные машины. Подчеркнем, что использование термина "оракульная машина" почти совпадает с его стандартным использованием. Незначительное отличие состоит в том, что оракульные машины, которые мы рассматриваем, используют в качестве оракула сохраняющую длину функцию вместо булевой функции (что более стандартно в теории сложности). Более того, мы предполагаем, что на входе 1^n оракульная машина делает запросы только длины $l(n)$. Такие соглашения не являются принципиальными, а просто позволяют несколько упростить изложение. Мы обозначаем как M^f оракульную машину M, которой дан доступ к оракулу f.

Определение 4.4 (Псевдослучайный ансамбль функций). : Ансамбль l-битовых функций называется **псевдослучайным**, если для любой вероятностной полиномиальной оракульной машины M, для любого полинома $p(\cdot)$ и всех достаточно больших n,

$$|\Pr[M^{F_n}(1^n) = 1] - \Pr[M^{H_n}(1^n) = 1]| < \frac{1}{p(n)}$$

где $H = \{H_n\}_{n\in\mathbb{N}}$ это равномерный ансамбль l-битовых функций.

Чтобы ансамбль функций имел практический смысл, мы требуем, чтобы псевдослучайные функции могли быть эффективно вычислены. То есть функции в ансамбле должны иметь короткие представления, которые позволяют их выбирать и вычислять. Эти свойства закладываются в следующее определение, в котором I это алгоритм, выбирающий представления функций (которые связываются с самими функциями при помощи отображения ϕ).

Определение 4.5 (Эффективно вычислимые ансамбли функций). Ансамбль l-битовых функций $F = \{F_n\}_{n\in\mathbb{N}}$ называется **эффективно вычислимым**, если выполняются следующие два условия:

1. Эффективная индексация: Существует вероятностный полиномиальный алгоритм I и отображение из строк в функции ϕ, такие что $\phi(I(1^n))$ и F_n распределены одинаково.

   Мы будем обозначать как f_i функцию, соответствующую строке i (то есть $f_i \stackrel{def}{=} \phi(i)$).

2. Эффективное вычисление: Существует полиномиальный алгоритм V, такой что $V(i,x) = f_i(x)$ для любых i в образе $I(1^n)$ и $x \in \{0,1\}^{l(n)}$.

В частности, из приведенного определения следует, что функции из эффективно вычислимого псевдослучайного ансамбля имеют относительно короткие представления (полиномиального, а не экспоненциального по n размера). Следовательно эффективно вычислимые ансамбли функций могут содержать только экспоненциальное количество функций (при том, что в предположении $l(n) = n$ всего функций дважды экспоненциальное количество).

Еще один момент, на который стоит обратить внимание, это то, что эффективно вычислимые псевдослучайные функции могут быть эффективно вычислены в данных точках *при условии, что описания функций также даны*. Однако, если функция (или ее описание) *неизвестно*, то значение функции в данной точке не может быть вычислено приближенно, даже в очень слабом смысле и даже если значения функции в других точках известны.

Терминология. В дальнейшем мы будем рассматривать только эффективно вычислимые ансамбли функций. Таким образом, если мы говорим о псевдослучайных функциях, то на самом деле имеются в виду функции, выбранные случайным образом из эффективно вычислимого псевдослучайного ансамбля.

4.4 Конструкция

Используя псевдослучайный генератор, мы можем построить псевдослучайный ансамбль функций (для $l(n) = n$), который является эффективно вычислимым.

Конструкция: Пусть G это детерминированный алгоритм, который преобразует входы длины n в строки длины $2n$. Обозначим за $G_0(s)$ $|s|$-битный префикс $G(s)$, а за $G_1(s)$ — $|s|$-битный суффикс $G(s)$ (то есть $G(s) = G_0(s)G_1(s)$). Для всех $s \in \{0,1\}^n$ мы определяем функцию $f_s\colon \{0,1\}^n \to \{0,1\}^n$, такую что для всех $\sigma_1, \ldots, \sigma_n \in \{0,1\}$,

$$f_s(\sigma_1\sigma_2\cdots\sigma_n) \stackrel{def}{=} G_{\sigma_n}(\cdots(G_{\sigma_2}(G_{\sigma_1}(s)))\cdots)$$

Пусть F_n это случайная величина, получаемая следующим образом: случайно выбираем $s \in \{0,1\}^n$ и полагаем $F_n = f_s$. Тогда пусть $F = \{F_n\}_{n\in\mathbb{N}}$ будет нашим ансамблем функций.

Получается, что значения функции f_s можно определять, проходя по путями длины n от корня к листьям в полном двоичном дереве глубины n, имеющем метки в вершинах. Корень дерева, который мы в дальнейшем будем называть вершиной уровня 0 в дереве имеет в качестве метки строку s. Если внутренняя вершина имеет метку r, то

ее левый сын имеет метку $G_0(r)$, а правый — метку $G_1(r)$. Значение $f_s(x)$ это строка, которая находится в листе дерева, достижимом от корня по пути, соответствующем строке x. Случайная переменная F_n распределена на множестве помеченных деревьев, соответствующих всем 2^n возможным пометкам корня, равномерным образом.

Функция, определенная на n-битовых строках построенного ансамбля, может описана при помощи n битов. Таким образом, выбор, замена и хранение такой функции может быть реализовано путем выбора, замены и хранения одного n-битового числа.

Теорема 4.2. *Пусть G и F такие, как в Конструкции, и предположим, что G это псевдослучайный генератор. Тогда F это эффективно вычислимый ансамбль псевдослучайных функций.*

Поскольку из существования односторонних функций следует существование псевдослучайного генератора, то мы сразу получаем следующее следствие:

Следствие 4.1. *Если существуют односторонние функции, то существуют и псевдослучайные функции.*

Доказательство. Очевидно, что ансамбль F является эффективно вычислимым. Чтобы доказать, что F является псевдослучайным, мы используем гибридный метод. В качестве k-ого гибрида мы возьмем функцию, которая получается при равномерном выборе меток для вершин k-ого (сверху) уровня дерева и вычислении меток для нижних уровней, как это делается в Конструкции 4. Гибрид номер 0 будет соответствовать случайной величине F_n (поскольку корню присваивается случайная метка), в то время как гибрид номер n будет соответствовать равномерно распределенной случайной величине H_n (поскольку каждому листу в таком случае присваивается равномерно распределенная метка). Ниже будет показано, что эффективная оракульная машина, которая различает соседние гибриды, может быть преобразована в алгоритм, который различает полиномиальное количество сэмплов $G(U_n)$ от полиномиального количества сэмплов U_{2n}. Используя теорему 5??????, мы получим противоречие с исходной гипотезой, состоящей в том, что G это псевдослучайный генератор. Детали будут изложены ниже.

Для всех k, где $0 \le k \le n$, мы определяем гибридное распределение H_n^k на множестве функций $f\colon \{0,1\}^n \to \{0,1\}^n$, как будет описано ниже. Для всех $s_1, s_2, \ldots, s_{2^k} \in \{0,1\}^n$, мы определяем функцию $f_{s_1,s_2,\ldots,s_{2^k}} : \{0,1\}^n \to \{0,1\}^n$, такую что

$$f_{s_1,s_2,\ldots,s_{2^k}}(\sigma_1,\sigma_2,\ldots,\sigma_n) \stackrel{def}{=} G_{\sigma_n}(\cdots G_{\sigma_{k+2}}(G_{\sigma_{k+1}}(s_{idx(\sigma_k\cdots\sigma_1)}))\cdots)$$

где $idx(\alpha)$ это индекс α в стандартном лексикографическом порядке двоиных строк длины $|\alpha|$. Получается, что $f_{s_1,s_2,\ldots,s_{2^k}}(x)$ вычисляется следующим образом: сначала префикс x длины k используется для выбора одного из s_j, а затем суффикс длины $n-k$ используется для определения того, какие из функций G_0 и G_1 применять на оставшихся уровнях (как в Конструкции 4). Случайная переменная H_n^k равномерно распределена на множестве всех $(2^n)^{2^k}$ возможных функций (соответствующих всем возможным выборам $s_1, s_2, \ldots, s_{2^k} \in \{0,1\}^n$). А именно:

$$H_n^k \stackrel{def}{=} f_{U_n^{(1)},\ldots,U_n^{(2^k)}}$$

где $U_n^{(j)}$ это независимые случайные величины, которые равномерно распределены на множестве $\{0,1\}^n$.

Теперь понятно, что H_n^0 совпадает с F_n, в то время как H_n^n совпадает с H_n. Снова, как это обычно бывает при использовании гибридного метода, возможность отличить крайние гибриды дает возможность отличить пару соседних гибридов. Эта возможность приводит к противоречию с псевдослучайностью G. Ниже будет изложено почему.

Будем доказывать от противного, то есть предположим, что ансамбль функций F не псевдослучаен. Следовательно существует вероятностная полиномиальная машина M и полином p, такой что для бесконечно большого количества n

$$\Delta(n) \stackrel{def}{=} |\Pr[M^{F_n}(1^n) = 1] - \Pr[M^{H_n}(1^n) = 1]| > \frac{1}{p(n)}$$

Пусть $t(\cdot)$ это полином, ограничивающий время работы $M(1^n)$ (такой полином существует, поскольку M это полиномиальная машина). Следовательно на входе 1^n оракульная машина M делает не более $t(n)$ запросов к оракулу (поскольку количество запросов очевидно ограничено временем работы). Используя машину M, мы построим алгоритм D, который будет отличать $t(\cdot)$-результаты ансамбля $\{G(U_n)\}_{n\in\mathbb{N}}$ от $t(\cdot)$-результатов ансамбля $\{U_{2n}\}_{n\in\mathbb{N}}$ так, как будет описано ниже.

Алгоритм D: На входе $\alpha_1, \alpha_2, \ldots, \alpha_t \in \{0,1\}^{2n}$ (здесь $t = t(n)$) алгоритм D работает следующим образом. Сначала D выбирает равномерно распределенную величину $k \in \{0, 1, \ldots, n-1\}$. Этот случайный выбор, который мы будем в дальнейшем называть *контрольной точкой* это единственный случайный выбор, который делает алгоритм D. Затем алгоритм D запускает оракульную машину M (на входе 1^n) и отвечает на запросы M к оракулу, как будет описано далее. На первый запрос машины M, который мы обозначим как q_1, ответом является

$$G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(P_{\sigma_{k+1}}(\alpha_1)))\cdots)$$

где $q_1 = \sigma_1, \cdots \sigma_n$, ($\alpha_1$ это первая строка из входа), а $P_0(\alpha)$ (и соответственно $P_1(\alpha)$) это обозначения для n-битового префикса α (и соответственно n-битового суффикса α). Алгоритм D дополнительно записывает этот запрос (т.е. q_1). Ответ на каждый последующий запрос строится следующим образом: сначала проверяется, не равен ли его k-битовый префикс k-битовому префиксу одного из предыдущих запросов. В случае, если k-битовый префикс текущего запроса, который мы обозначаем как q_i, отличается от k-битового префикса всех предыдущих запросов, мы ассоциируем этот префикс с новой строкой их входа (т.е. α_i). Соответственно мы отвечаем на запрос q_i следующим образом:

$$G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(P_{\sigma_{k+1}}(\alpha_i)))\cdots)$$

где $q_i = \sigma_1, \cdots, \sigma_n$. Алгоритм D дополнительно записывает текущий запрос (т.е. q_i). Оставшийся случай это когда k-битовый префикс i-го запроса совпадает с k-битовым префиксом j-го запроса (по предположению $j < i$). Тогда мы записываем текущий запрос (т.е. q_i), но отвечаем на него, используя строку, ассоциированную с запросом q_j (то есть входную строку α_j). Соответственно ответ на запрос q_i выглядит так:

$$G_{\sigma_n}(\cdots(G_{\sigma_{k+2}}(P_{\sigma_{k+1}}(\alpha_j)))\cdots)$$

где $q_i = \sigma_1, \cdots, \sigma_n$. Наконец, когда машина M останавливается, алгоритм D тоже останавливается и выдат то же, что выдает машина M.

Если рассматривать этот процесс на двоичном дереве, которое было описано ранее, то алгоритм D отвечает на первый запрос, сначала помещая две половины α_1 в соответствующих детей вершины дерева, которая достижима по пути от корня, который соответствует $\sigma_1, \cdots, \sigma_k$. Пометки всех вершин поддерева, соответствующего $\sigma_1, \cdots, \sigma_k$ определяются пометками этих двух детей (как в конструкции F). Ответы на последующие запросы получаются путем прохода по соответствующим путям от корня. В случае, если путь не проходит через вершину $(k+1)$-го уровня, у которой уже есть пометка, мы присваиваем этой вершине и ее брату новую строку (взятую из входа). Для простоты в случае, если i-ый запрос требует новую строку, мы используем i-ую строку из входа (вместо того, чтобы использовать первую из неиспользованных строк). В случае, если путь для нового запроса проходит через вершину $(k+1)$-го уровня, которая уже была помечена, мы используем эту метку для вычисления последующих меток вдоль пути (и в частности метку листа). Подчеркнем, что алгоритм *не* вычисляет метки во всех вершинах поддерева, соответствующего $\sigma_1, \cdots, \sigma_k$ (хотя эти метки и определяются меткой в вершине, соответствующей $\sigma_1, \cdots, \sigma_k$), а вместо этого вычисляет только метки в вершинах, которые находятся на путях, соответствующих запросам.

Очевидно, что алгоритм D может быть реализован за полиномиальное время. Осталось проверить, правильно ли он работает. Ключевым замечанием здесь является соответствие между действиями алгоритма D в контрольной точке k и гибридами номер k и $k+1$:

- Когда входы берутся из $t(n)$-результата U_{2n} (и алгоритм D выбирает k в качестве контрольной точки), запускаемая машина M ведёт себя в точности так, как это делает $k+1$-ый гибрид. Это так, поскольку D помещает две половины истинно случайных $2n$-битовых строк на уровень $k+1$ (что то же самое, что поместить истинно случайные n-битовые строки на весь $k+1$-ый уровень).

- С другой стороны, когда входы берутся из $t(n)$-результата $G(U_n)$ (и алгоритм D выбирает k в качестве контрольной точки), то M ведет себя точно так же, как на k-ом гибриде. Действительно, D не помещает (неизвестные ему) исходные строки (на основе которых генерируются псевдослучайные строки) на уровень k; однако помещение двух половин псевдослучайных строк на уровень $k+1$ даёт тот же самый эффект.

Таким образом:

Утверждение 4.3. *Пусть n это целое число, а $t \stackrel{def}{=} t(n)$. Пусть K это случайная переменная, которая задает случайный выбор контрольной точки алгоритмом D (на входной последовательности длины t из $2n$-битовых строк). Тогда для всех $k \in \{0, 1, \ldots, n-1\}$,*

$$\Pr[D(G(U_n^{(1)}), \ldots, G(U_n^{(t)})) = 1 | K = k] = \Pr[M^{H_n^k}(1^n) = 1]$$

$$\Pr[D(U_{2n}^{(1)}, \ldots, U_{2n}^{(t)})) = 1 | K = k] = \Pr[M^{H_n^{k+1}}(1^n) = 1]$$

где $U_n^{(i)}$ *и* $U_{2n}^{(j)}$ *это независимые случайные величины, равномерно распределенные на* $\{0,1\}^n$ *и* $\{0,1\}^{2n}$ *соответственно.*

Доказательство. Рассмотрим альтернативное определение величины H_n^m для всех $0 \leq m \leq n$. Альтернативное определение будет похоже на то, как алгоритм D отвечает на запросы оракульной машины M (мы будем использовать здесь обозначение m вместо k, поскольку m *не обязательно* совпадает с контрольной точкой k, выбранной алгоритмом D). Это определение состоит из переключения двух случайных процессов, которые сначала одновременно выбирают функцию $g\colon \{0,1\}^m \to \{0,1\}^n$, которая в дальнейшем используется для определения функции $f\colon \{0,1\}^n \to \{0,1\}^n$. Первый случайный процесс, который мы обозначим за ρ, это произвольный процесс ("данный нам снаружи"), который определяет точки в образе g (процесс ρ соответствует запросам машины M, в то время как второй процесс соответствует тому, как алгоритм A отвечает на эти запросы). Второй процесс, который мы обозначим как ψ, присваивает равномерным образом выбранные n-битовые строки каждой точке, выбранной процессом ρ, таким образом задавая значение функции g в этой точке. Подчеркнем, что в случае, если ρ выбирает старую точку (ту, которую он уже выбирал), второй процесс ничего не делает (т.е. оставляет значение функции g в этой точке неизменным). Процесс ρ может зависеть от истории двух процессов и в частности от значений, которые были выбраны для предыдущих точек. Когда ρ завершается, второй процесс ψ выбирает случайные значения для оставшихся точек, в которых значения не определены (если они есть). Подчеркнем, что второй процесс ψ фиксирован для всех возможных выборов первого процесса ρ.

Когда функция $g\colon \{0,1\}^m \to \{0,1\}^n$ полностью определена, мы определяем функцию $f^g\colon \{0,1\}^n \to \{0,1\}^n$ следующим образом:

$$f^g(\sigma_1, \sigma_2, \cdots, \sigma_n) \stackrel{def}{=} G_{\sigma_n}(\cdots(G_{\sigma_{m+2}}(G_{\sigma_{m+1}}(g(\sigma_m, \cdots, \sigma_1))))\cdots)$$

Читатель может с легкостью убедиться в том, что f^g равняется $f_{g(0^m),\ldots,g(1^m)}$ (как это было определено в гибридной конструкции выше). Также легко проверить, что описанный случайный процесс (т.е. чередование ψ с любым ρ) выдает функцию g, которая равномерно распределена на множестве всех возможных функций, отображающих m-битовые строки в n-битовые строки. Следовательно описанный случайный процесс приводит к результату (т.е. функции), который распределен так же, как случайная переменная H_n^m. Предположим теперь, что контрольная точка, выбранная D равна k и что входы D независимо и равномерно выбираются из $\{0,1\}^{2n}$. В таком случае то, как D отвечает на запросы M, можно рассматривать как размещение независимо и равномерно выбранных n-битовых строчек в качестве меток вершин $k+1$-го уровня. Следовательно, то, как D отвечает на запросы M, соответствует описанному выше процессу при $m = k+1$ (где M играет роль ρ, а A играет роль ψ). Таким образом, в этом случае M запускается с доступом к случайной переменной $k+1$-го гибрида.

Предположим, с другой стороны, что (снова контрольная точка, выбранная алгоритмом D равная k) входы D независимо выбираются так, что каждый из них распределен так же, как и $G(U_n)$. В таком случае то, каким образом D отвечает на запросы M, можно рассматривать как размещение независимо и равномерно распределенных n-битовых

строк в качестве меток вершин k-го уровня. Следовательно то, как D отвечает на запросы M, соответсвует описанному выше процессу пр $m = k$. Таким образом, в этом случае M запускается с доступом к случайной переменной k-го □

Используя утверждение 4.3 и вводя обозначение

$$\Delta(n) = \Pr[M^{H_n^0}(1^n) = 1] - \Pr[M^{H_n^k}(1^n) = 1]$$

получаем:

$$\Pr[D(G(U_n^{(1)}), \ldots, G(U_n^{(t)})) = 1] - \Pr[D(U_{2n}^{(1)}, \ldots, U_{2n}^{(t)}) = 1]$$
$$= (\frac{1}{n}\sum_{k=0}^{n-1} \Pr[M^{H_n^k}(1^n) = 1]) - (\frac{1}{n}\sum_{k=0}^{n-1} \Pr[M^{H_n^{k+1}}(1^n) = 1]) = \frac{\Delta(n)}{n}$$

что, по предположению от противного, больше, чем $\frac{1}{n*p(n)}$ для бесконечного числа n. Следовательно D (вероятностный полиномиальный алгоритм) отличает полиномиальное количество сэмплов $G(U_n)$ от полиномиального количества сэмплов U_{2n}. Используя теорему 4.1, мы получаем противоречие с предположением теоремы о том, что G это псевдослучайный генератор, таким образом теорема доказана. □