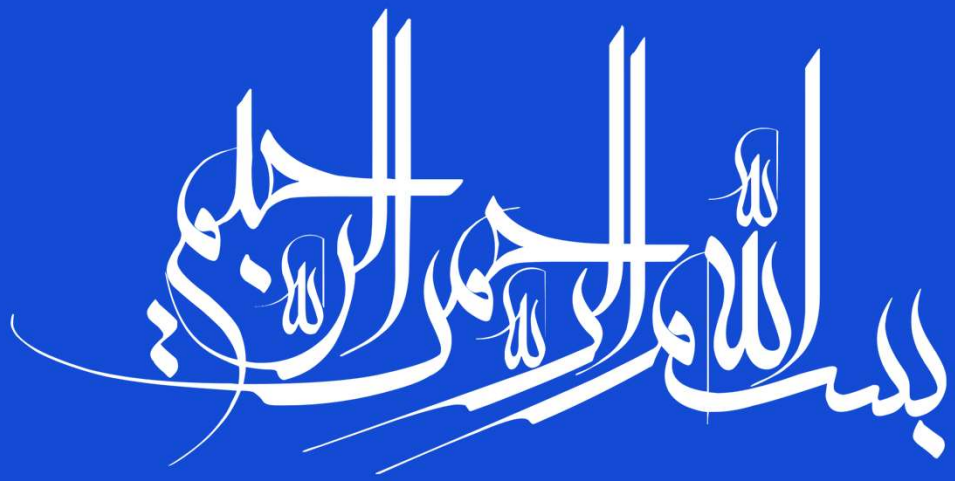

بسم الله الرحمن الرحيم

عنوان : ارائه فصل هشتم

نام و نام خانوادگی:

سیدساسان میرطاهری

تاریخ ارائه:

24 آذر 1404

عنوان : ارائه فصل هشتم

عنوان : بخش 8.4.2 تا آخر فصل

رشته: رشته شبکه های کامپیوتری

استاد : جناب آقای دکتر لاجوردی

گرایش: کامپیوتر

استاد محترم و دانشجو

استاد مربوطه

دکتر لاجوردی

معاون فناوری اطلاعات دانشگاه

جامع انقلاب اسلامی

دانشجو

سید ساسان میرطاهری

فهرست مطالب ارائه

بخش 8.4.2
Increasing the Expansion Factor

بخش 8.5
Constructing Pseudorandom Functions

بخش 8.6
(Strong) Pseudorandom Permutations

بخش 8.7
فرضیات لازم برای رمزنگاری کلید خصوصی

بخش 8.8
Computational Indistinguishability

8.4.2 – افزایش ضریب بسط (Increasing the Expansion Factor)

در این بخش نویسندگان نشان می‌دهند که اگر یک تولیدکننده شبه‌تصادفی (PRGداشته باشیم که فقط **یک بیت بیشتر** از ورودی تولید کند (یعنی ورودی nبیت، خروجی n+1بیت)، می‌توانیم از آن برای ساختن تولیدکننده‌ای استفاده کنیم که **به اندازه دلخواه (به‌صورت چندجمله‌ای)** خروجی ایجاد کند.

به زبان ساده:

اگر بتوانیم از یک ورودی nبیتی، n+1بیت شبه‌تصادفی بسازیم، می‌توانیم این فرآیند را طوری زنجیره کنیم که از همان ورودی **مثلاً n+100**بیت، یا **n + poly(n)**بیت خروجی تولید شود

قضیه 8.19

اگر یک تولیدکننده شبه‌تصادفی G با ضریب بسط $n+1$ وجود داشته باشد، آنگاه برای هر چندجمله‌ای poly، تولیدکننده‌ای به نام $\hat{G}$ وجود دارد که ضریب بسط آن $poly(n)$ است.

ایده کلی اثبات

برای شروع، حالت ساده را در نظر می‌گیریم: می‌خواهیم تولیدکننده‌ای بسازیم که **$n+2$ بیت** خروجی بدهد.

روش کار $\hat{G}$:

1. یک seed ورودی s در نظر بگیر. $G(s)$ را اجرا کن تا $n+1$ بیت به دست آید.
2. n بیت اول خروجی اول را دوباره به‌عنوان seed جدید وارد G کن.
3. بیت آخر خروجی اول را هم به انتها اضافه کن.

به این ترتیب خروجی جدید $n+2$ بیت می‌شود.

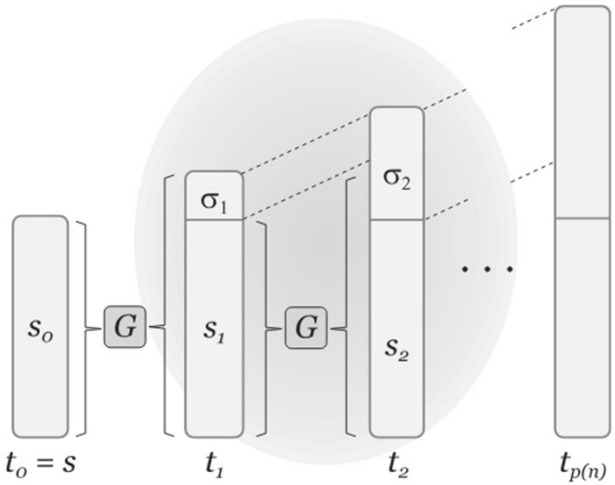
این فرآیند در **شکل 8.1** نشان داده شده است.

نکته مهم این است که:

بار دوم که G اجرا می‌شود، ورودی آن دیگر یک seed تصادفی واقعی نیست بلکه یک seed **شبه‌تصادفی** است.

اثبات نشان می‌دهد که این موضوع آسیبی به امنیت یا شبه‌تصادفی بودن خروجی نمی‌زند.

ساخت توزیع‌ها برای اثبات



1

- **H0**
- یک t_0 تصادفی n بیتی انتخاب می‌شود.
- خروجی $\hat{G}(t_0)$ تولید می‌شود.

2

- **H1**
- یک رشته تصادفی n+1 بیتی انتخاب می‌شود.
- n بیت اول آن به‌عنوان ورودی G استفاده می‌شود و بیت آخر همان آخر خروجی است.

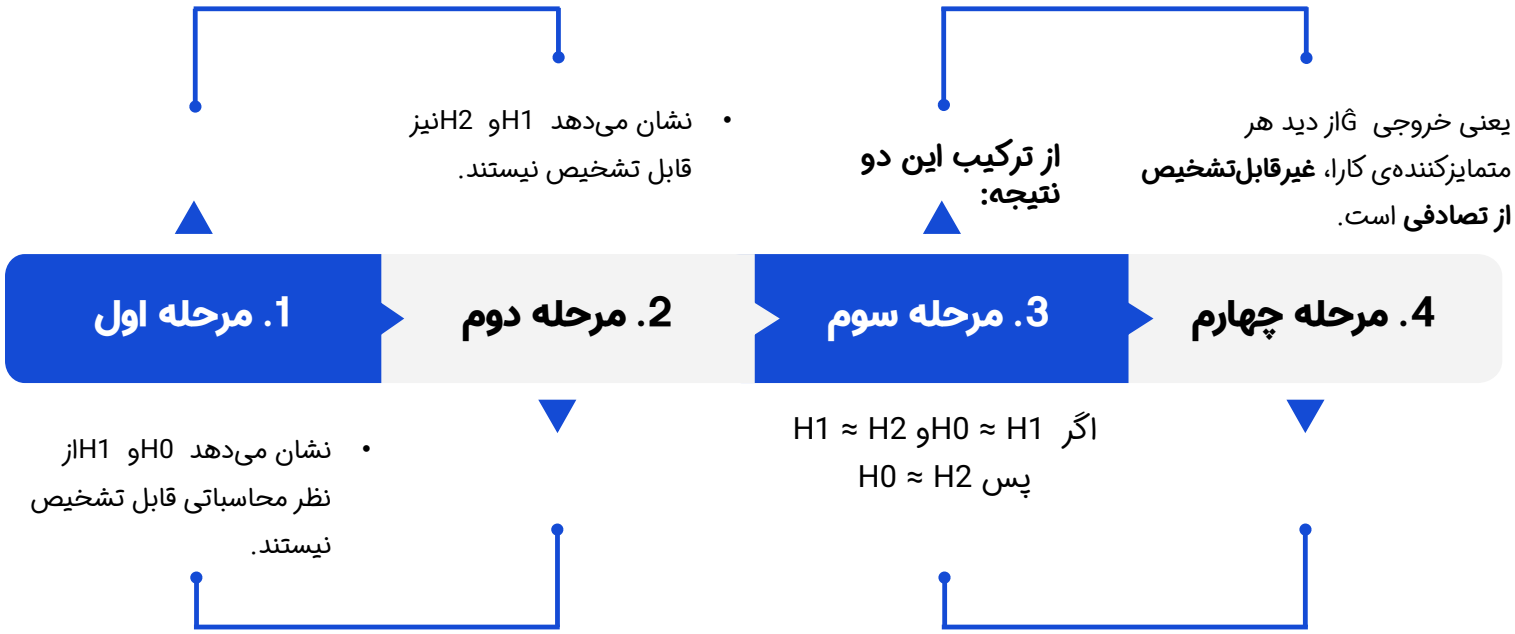
3

- **H2**
- یک رشته کاملاً تصادفی n+2 بیتی است.

- هدف اثبات این است که هیچ الگوریتم کارای تمایزدهنده‌ای نتواند **H0** را از **H2** تشخیص دهد؛ یعنی خروجی $\hat{G}$ شبه‌تصادفی است.

ایده اصلی اثبات با استدلال " Hybrid "

اثبات شامل دو مرحله است:



گسترش به حالت کلی p(n)

اگر بخواهیم خروجی p(n) + n بیت تولید کنیم، این فرآیند را p(n) بار تکرار می‌کنیم:

ابتدا $t_0 = s$

برای i از 1 تا p(n):

n بیت اول t_{i-1} وارد G می‌شود

تمام خروجی قبلی به انتهای خروجی جدید اضافه می‌شود

در نهایت $t_{p(n)}$ خروجی نهایی است.

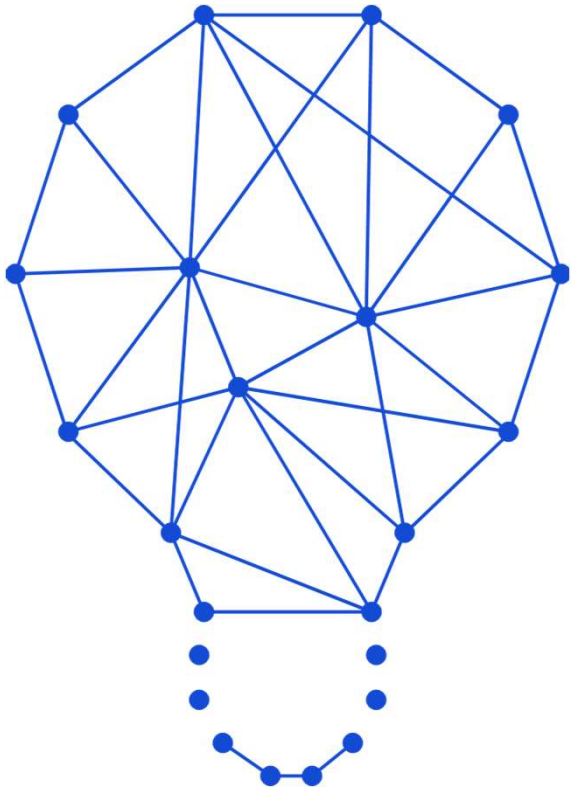
برای این بخش، نویسندگان یک **Hybrid Argument** عمومی معرفی می‌کنند:

در آن، توزیع‌های H_0 تا $H_{p(n)}$ تعریف شده‌اند؛ هر کدام نمایانگر مرحله‌ای هستند که در آن تعدادی از ورودی‌ها تصادفی هستند و بقیه شبه‌تصادفی.

نتیجه نهایی:

$\hat{G}$ یک تولیدکننده شبه‌تصادفی با بسط دلخواه است.

جمع‌بندی پایانی بخش 8.4.2



هدف اصلی

در نهایت، با ترکیب این روش با تابع یک‌طرفه f و predicate سخت hcکه

در بخش قبلی معرفی شد، یک PRGبا بسط دلخواه ساخته می‌شود:

اهداف فرعی

که در آن از fچند بار و از hcبرای تولید هر بیت استفاده می‌شود.

این تکنیک اساس بسیاری از ساختارهای PRGدر رمزنگاری مدرن است.

$$\hat{G}(s) = f^{(\ )}(s) \;||\; hc(f^{(\ -1)}(s)) \;||\; \ldots \;||\; hc(s)$$

8.5 – ساخت توابع شبه‌تصادفی (Constructing Pseudorandom Functions)

در این بخش نشان داده می‌شود که چگونه می‌توان از یک تولیدکننده شبه‌تصادفی که خروجی‌اش دو برابر ورودی است (**length-doubling PRG**) ، یک **تابع شبه‌تصادفی** ساخت.

هدف، تعریف تابعی است که ورودی n بیتی را به خروجی n بیتی نگاشت کند، طوری که برای هر کلید k، این تابع F_k ظاهری کاملاً تصادفی داشته باشد.

ایده اصلی

اگر یک تابع Fشبه‌تصادفی باشد، باید ورودی‌های nبیتی را به خروجی‌های nبیتی طوری نگاشت کند که:

- برای هر کلید k، تابع F_k(s)از دید یک متمایزکننده کارا تقریباً تصادفی باشد.
- محاسبه F_k(x)کارآمد باشد.
- نسبت به یک تابع کاملاً تصادفی روی 2^ nورودی، رفتار مشابهی داشته باشد.

پیچیدگی:

تابع تصادفی روی nبیت، دارای 2^ nخروجی nبیتی است.

اگر بخواهیم این را با یک PRGبسازیم، نباید مجبور شویم **کل خروجی را به طور کامل بسازیم** (که نمایی است). باید بتوانیم مقدار F_k(x)را بدون محاسبه همه چیز به دست آوریم.

نمونه ساده با ورودی ۲ بیتی

فرض کنید G یک PRG با ضریب بسط $2n$ است:

$G(k) = G_0(k) \,||\, G_1(k)$

که دو نصف خروجی را تشکیل می‌دهد. هرکدام n بیت.

نویسندگان برای ورودی‌های ۲ بیتی، تابع زیر را تعریف می‌کنند:

$F\_k(00) = G_0(G_0(k))$

$F\_k(01) = G_1(G_0(k))$

$F\_k(10) = G_0(G_1(k))$

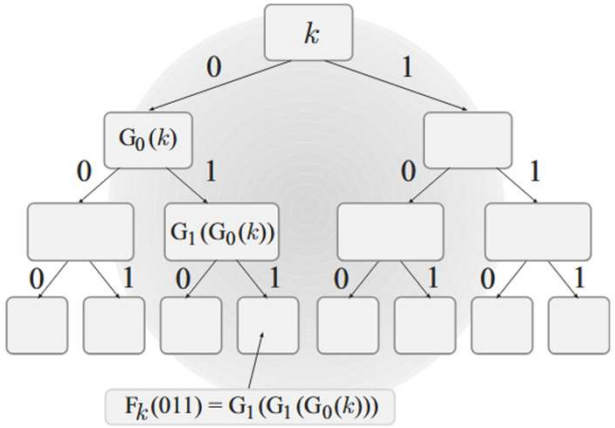
$F\_k(11) = G_1(G_1(k))$

$$F_k(00) = G_0(G_0(k)) \qquad F_k(10) = G_0(G_1(k))$$
$$F_k(01) = G_1(G_0(k)) \qquad F_k(11) = G_1(G_1(k)).$$

ایده:

ابتدا کلید k را وارد PRG می‌کنیم و آن دو خروجی را $G_0(k)$ و $G_1(k)$ می‌نامیم؛ سپس بسته به بیت‌های ورودی، دوباره یکی از این دو را وارد G می‌کنیم.

شبه‌تصادفی بودن



چرا این چهار مقدار شبه‌تصادفی هستند؟

- چون G(k) شبه‌تصادفی است، پس $G_0(k)$ و $G_1(k)$ مثل دو مقدار تصادفی‌اند.
- اگر آن‌ها را وارد G کنیم:

$G(G_0(k))$ و $G(G_1(k))$ نیز شبه‌تصادفی‌اند.

- پس هر چهار مقدار، غیرقابل‌تشخیص از ۴ خروجی تصادفی n بیتی هستند.

برای اثبات ر

$$G_0(G_0(k)) \parallel G_1(G_0(k)) \parallel G_0(G_1(k)) \parallel G_1(G_1(k))$$

$$G_0(k_0) \parallel G_1(k_0) \parallel G_0(k_1) \parallel G_1(k_1) = G(k_0) \parallel G(k_1).$$

گسترش به ورودی‌های n بیتی

ایده قبلی را می‌توان به ورودی‌های nبیتی تعمیم داد:

هر ورودی xرا مانند رشته‌ای با nبیت در نظر می‌گیریم:

$x = x_1\ x_2\ \dots\ x_n$

سپس از kشروع می‌کنیم و هر بار با توجه به بیت بعدی، یکی از دو شاخه G_0 یا G_1را اعمال می‌کنیم:

$F_k(x) = G_{x_n}(\dots G_{x_2}(G_{x_1}(k)) \dots)$

این ساختار مانند **پیمایش یک درخت دودویی کامل با عمق n** است:

- ریشه: مقدار k
- هر سطح : اعمال G_0 یا G_1
- برگ : خروجی F_kبرای یک ورودی خاص

در شکل 8.2 ارائه شده است (درخت دودویی با عمق 3).

مزیت مهم

اگر بخواهیم مقدار $F_k(x)$را حساب کنیم:

- نیازی نیست کل درخت را بسازیم.
- فقط کافی است nبار تابع Gاجرا شود.

با اینکه درخت 2^nبرگ دارد، محاسبه مقدار یک برگ فقط O(n) هزینه دارد.

$$F_k(x) = G_{x_n}(\cdots G_{x_1}(k)\cdots),$$

صورت رسمی ساخت (Construction 8.20)

تابع شبه‌تصادفی F به شکل زیر تعریف می‌شود:

$$F_k(x_1\ x_2\ \dots\ x_n) = G_{x_n}(\dots(G_{x_2}(G_{x_1}(k)))\dots)$$

که در آن:

- G خروجی خود را به دو قسمت تقسیم می‌کند:
  - $G_0(v)$: نصف اول
  - $G_1(v)$: نصف دوم

$$F_k(x_1x_2\cdots x_n) = G_{x_n}\left(\cdots(G_{x_2}(G_{x_1}(k)))\cdots\right).$$

قضیه 8.21: این ساخت یک تابع شبه‌تصادفی است

بیان:
اگر Gیک تولیدکننده شبه‌تصادفی با ضریب بسط 2 nباشد، آنگاه تابع Fحاصل از Construction 8.20یک **تابع شبه‌تصادفی صحیح و امن** است.

ایده اثبات

اثبات در دو مرحله است:

مرحله اول: ترکیب tخروجی PRGهمچنان شبه‌تصادفی است

نشان داده می‌شود که:
اگر به جای یک خروجی G، تعداد t(n)خروجی Gرا کنار هم بگذاریم، این بلوک بزرگتر هنوز از تصادفی قابل‌تشخیص نیست.
این بخش با یک روش hybridانجام می‌شود.

مرحله دوم: ساخت PRF

درختی با عمق nدر نظر بگیرید. در سطح iیک مقدار نود به طور تصادفی (منطبق با مدل Hybrid)انتخاب می‌شود.

توزیع‌های H_iتعریف می‌شوند:

- H_0: فقط ریشه تصادفی است (همان ساخت F_k)
- H_n: همه برگ‌ها تصادفی‌اند (یک تابع کاملاً تصادفی)

اگر بتوان H_iرا از H_{i+1}تشخیص داد، به معنای شکستن PRGاست.

ولی چون Gامن است:

$$H_0 \approx H_1 \approx H_2 \approx \ldots \approx H_n$$

در نتیجه خروجی F_kقابل تشخیص از یک تابع کاملاً تصادفی نیست.

این نشان می‌دهد Fیک **تابع شبه‌تصادفی** معتبر است.

جمع‌بندی بخش 8.5

- از یک PRG با خروجی دو برابر ورودی، می‌توان یک PRF ساخت.
- برای هر ورودی x، مسیر انتخابی از یک درخت دودویی کامل طی می‌شود.
- امنیت از طریق Hybrid Argument ثابت می‌شود.
- PRFها پایه بسیاری از سازه‌های رمزنگاری مثل بلاک‌سایفر، MAC و ... هستند.

8.6 – ساخت جایگشت‌های شبه‌تصادفی Strong Pseudorandom Permutations

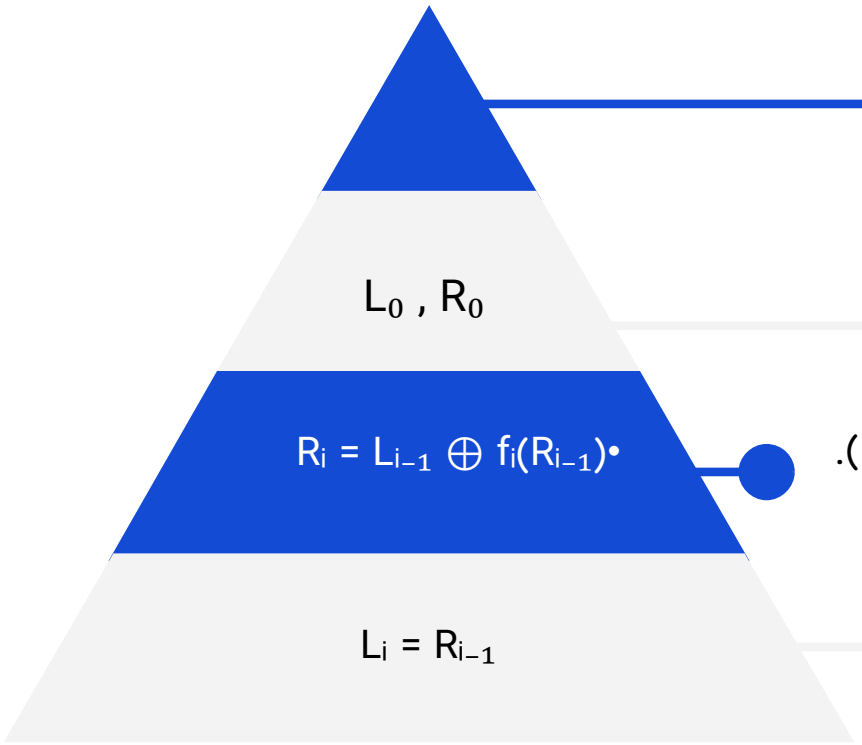
هدف بخش

در این بخش نشان داده می‌شود:

- چگونه می‌توان با استفاده از یک **تابع شبه‌تصادفی**، یک **جایگشت شبه‌تصادفی (PRP)** ساخت.
- استفاده از ساختار معروف **شبکه فایستل (Feistel Network)** برای ایجاد یک تابع قابل وارون‌سازی.
- این‌که با تعداد راندهای مناسب، شبکه فایستل می‌تواند:
- با **۳ راند PRP** → بسازد
- با **۴ راند SPRP** → (جایگشت شبه‌تصادفی قوی) بسازد

نکته اول

شبکه فایستل (Feistel Network)



شبکه فایستل یک روش استاندارد برای تبدیل هر تابع به یک **جایگشت قابل وارون‌سازی** است.

یک ورودی ۲ nبیتی به دو نیمه nبیتی تقسیم می‌شود:
در هر راند:

که f_i یک تابع $n \to n$ دلخواه است (لازم نیست وارون‌پذیر باشد).
نتیجه مهم:

شبکه فایستل همیشه وارون‌پذیر است.

این ویژگی باعث شده اساس بسیاری از بلاک‌سایفرها مثل DESباشد.

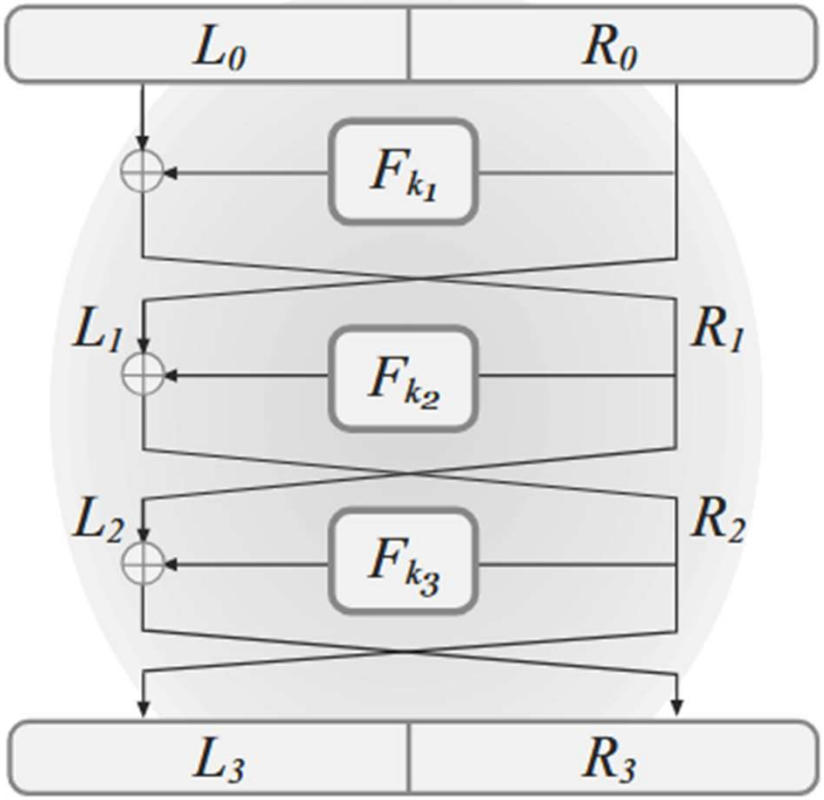
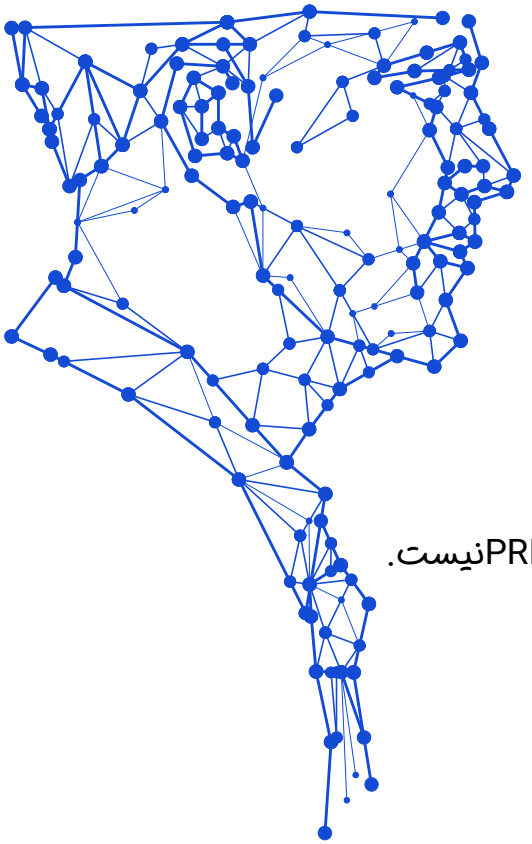


FIGURE 8.3: A three-round Feistel network, as used to construct a pseudorandom permutation from a pseudorandom function.

ساخت اولین PRP اما نا امن



فرض کنید Fیک تابع شبه‌تصادفی باشد.

تعریف ساده اول: فقط یک راند فایستل با F:

$F^{(1)}_k(x) = Feistel\_Fk(x)$

اما این ساخت **ناامن** است.

چرا؟

در یک راند فایستل:

خروجی نصف اول = نصف دوم ورودی است $L_1 = R_0$

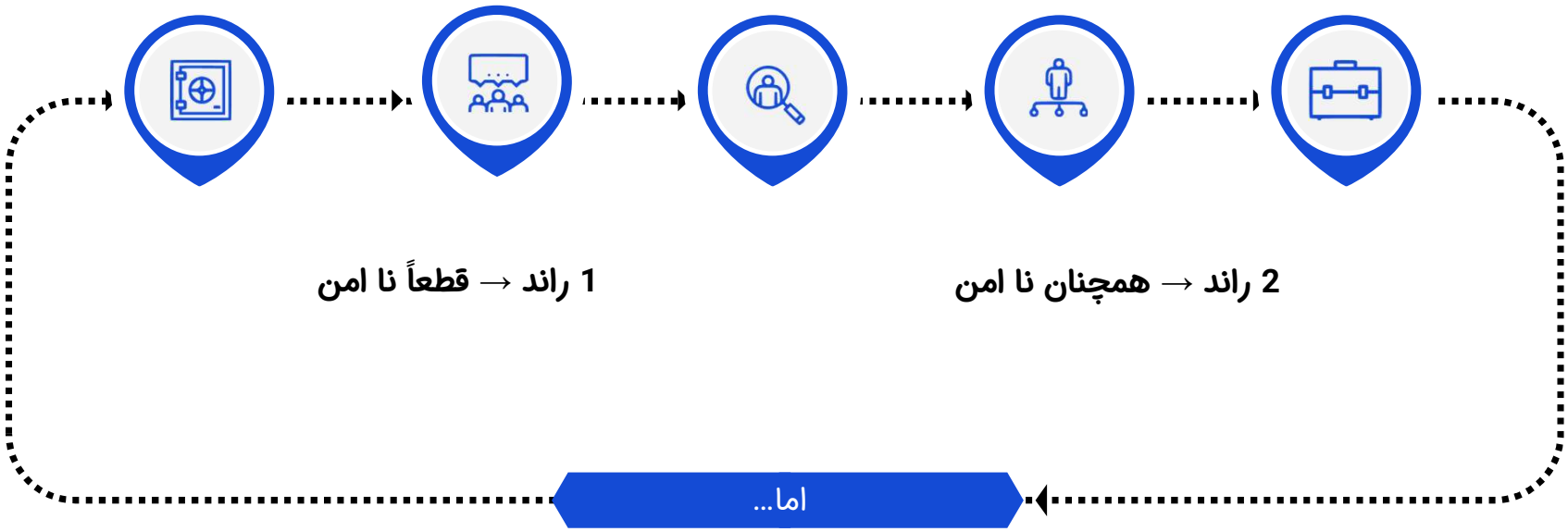
در یک جایگشت تصادفی چنین رابطه‌ای وجود ندارد.

بنابراین یک مهاجم می‌تواند خیلی راحت این الگو را کشف کند ← پس این ساخت PRPنیست.

ساخت دو راندی Two-round Feistel

این حالت نیز **شبه‌تصادفی نیست**.

بنابراین:



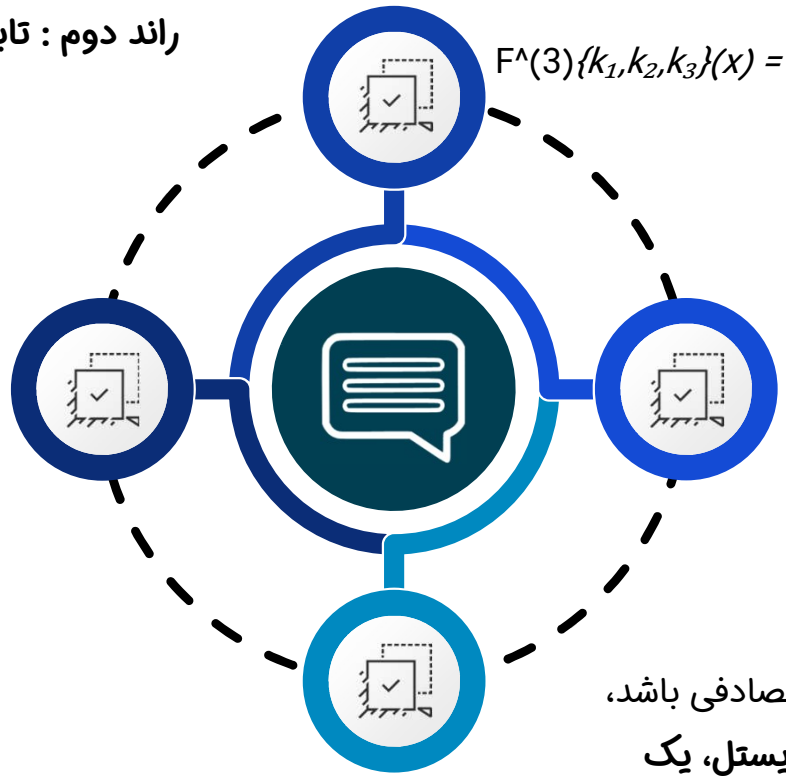
قضیه بسیار مهم: سه راند کافی است

ساخت 3 راندی: F^(3)

ایده:

F^(3)$\{k_1,k_2,k_3\}(x) = Feistel$(F_{k1}, F_{k2}, F_{k3})(x)

یعنی:



راند اول : تابع F با کلید k_1

راند دوم : تابع F با کلید k_2

راند سوم : تابع F با کلید k_3

قضیه 8.22

اگر Fیک تابع شبه‌تصادفی باشد،

ساخت سه‌راندی فایستل، یک

جایگشت شبه‌تصادفی PRP است.

خلاصه‌ای از ایده اثبات (بسیار ساده‌شده برای ارائه)

در اثبات، فرض می‌کنیم سه تابع f_1، f_2، f_3 به‌جای PRFواقعی، توابع کاملاً تصادفی هستند.
سپس نشان داده می‌شود:

- احتمال برخورد collisionروی نیمه‌های میانی ناچیز است
- اگر برخورد وجود نداشته باشد، خروجی فایستل مانند یک جایگشت تصادفی عمل می‌کند
- متمایزکننده نمی‌تواند تشخیص دهد سیستم واقعی از سیستم تصادفی است

اثبات از روش **Hybrid Argument** و تحلیل احتمالاتی استفاده می‌کند.
در نهایت ثابت می‌شود که با احتمال بالا، خروجی‌ها طوری توزیع می‌شوند که از یک جایگشت واقعی غیرقابل تمایز است.

این ساخت هنوز Strongنیست

فایستل سه‌راندی **PRP است اما SPRP نیست**.

یعنی:

- اگر مهاجم فقط به جلو دسترسی داشته باشد → امن است
- اگر مهاجم به تابع معکوس هم دسترسی داشته باشد → امن نیست

برای SPRPنیاز به یک راند اضافه داریم.

ساخت 4 راندی – Strong Pseudorandom Permutation

این ساخت در **Construction 8.23** ارائه شده است.

تعریف ساخت 4 راندی

ورودی:

- کلید k شامل چهار زیرکلید k_1, k_2, k_3, k_4
- ورودی xشامل L_0و R_0

CONSTRUCTION 8.23

Let F be a keyed, length-preserving function. Define the keyed permutation $F^{(4)}$ as follows:

- **Inputs:** A key $k = (k_1, k_2, k_3, k_4)$ with $|k_i| = n$, and an input $x \in \{0,1\}^{2n}$ parsed as (L_0, R_0) with $|L_0| = |R_0| = n$.
- **Computation:**
  1. Compute $L_1 := R_0$ and $R_1 := L_0 \oplus F_{k_1}(R_0)$.
  2. Compute $L_2 := R_1$ and $R_2 := L_1 \oplus F_{k_2}(R_1)$.
  3. Compute $L_3 := R_2$ and $R_3 := L_2 \oplus F_{k_3}(R_2)$.
  4. Compute $L_4 := R_3$ and $R_4 := L_3 \oplus F_{k_4}(R_3)$.
  5. Output (L_4, R_4).

محاسبات:

$L_1 = R_0.1$

$R_1 = L_0 \oplus F_{k_1}(R_0)$

$L_2 = R_1.2$

$R_2 = L_1 \oplus F_{k_2}(R_1)$

$L_3 = R_2.3$

$R_3 = L_2 \oplus F_{k_3}(R_2)$

$L_4 = R_3.4$

$R_4 = L_3 \oplus F_{k_4}(R_3)$

خروجی: L_4 , R_4

قضیه 8.24

اگر F یک PRF باشد،

این ساخت چهار راندی یک جایگشت شبه‌تصادفی قوی SPRP است.

این یعنی:

•هم در حالت عادی forward oracle

هم در حالت دسترسی به معکوس inverse oracle

قابل‌تشخیص ا

این دقیقاً همار

$$\left| \Pr_{w \leftarrow \{0,1\}^{2n}}[D(w) = 1] - \Pr_{s \leftarrow \{0,1\}^{n}}[D(G(s)) = 1] \right| \geq \Pr[\mathsf{Invert}_{\mathcal{A},G}(n) = 1] - 2^{-n}.$$

جمع‌بندی بخش 8.6

1.شبکه فایستل راهی عمومی برای ساختن یک تابع قابل معکوس‌سازی است.

2.اگر از PRF در راندهای فایستل استفاده کنیم:

1. ۳ راند → PRP امن
2. ۴ راند → SPRP امن (مانند بلاک‌سایفر واقعی)

3.این نتیجه اساس بسیاری از الگوریتم‌های رمزنگاری بلاکی است.

8.7 – فرضیات لازم برای رمزنگاری کلید خصوصی

•اگر یک **پرموتیشن یک‌طرفه** وجود داشته باشد → می‌توان یک **تولیدکننده شبه‌تصادفی (PRG)** ساخت.

•اگر PRG وجود داشته باشد → می‌توان **تابع شبه‌تصادفی (PRF)** ساخت.

اگر PRFوجود داشته باشد → می‌توان **جایگشت شبه‌تصادفی قوی SPRP**ساخت.

•**توابع یک‌طرفه هم لازم و هم کافی‌اند.**
•یعنی *حداقل* فرض لازم برای داشتن رمزنگاری کلید خصوصی همین یک فرض است.

PRG → وجود تابع یک‌طرفه

گزاره 8.27

اگر یک تولیدکننده شبه‌تصادفی وجود داشته باشد،

آنگاه توابع یک‌طرفه نیز وجود دارند.

ایده اثبات

فرض کنید G یک PRG با ضریب بسط 2n باشد.

می‌خواهیم نشان دهیم که خود G یک تابع یک‌طرفه است.

چرا؟

- اگر کسی بتواند G را معکوس کند، می‌تواند تشخیص دهد خروجی G شبه‌تصادفی نیست.
- چون اگر خروجی واقعاً از G آمده باشد، معکوسش وجود دارد؛

ولی اگر کاملاً تصادفی باشد، احتمال اینکه قابل معکوس‌سازی باشد بسیار کم است.

پس:

اگر معکوس کردن ممکن باشد → تمایز دادن ممکن می‌شود → خلاف تعریف PRG.

$$\left| \Pr_{w \leftarrow \{0,1\}^{2n}}[D(w) = 1] - \Pr_{s \leftarrow \{0,1\}^{n}}[D(G(s)) = 1] \right| \geq \Pr[\mathsf{Invert}_{\mathcal{A},G}(n) = 1] - 2^{-n}.$$

رمزنگاری خصوصی → تابع یک‌طرفه

نویسندگان سپس بررسی می‌کنند که آیا یک **سیستم رمزنگاری کلید خصوصی امن** لزوماً وجود یک تابع یک‌طرفه را نتیجه می‌دهد.

گزاره 8.28

اگر یک رمزنگاری کلید خصوصی وجود داشته باشد که:

- در برابر استراق سمع ایمن باشد ((EAV-secure
- پیام‌هایی به طول **n ۲ بیت** را با کلیدی به طول **n بیت** رمز کند

آنگاه **یک تابع یک‌طرفه وجود دارد**.

ایده اثبات

تابع f زیر را تعریف کنید:

f(k, m, r) = Enc_k(m ; r) || m

که در آن:

- k: کلید n بیتی
- m: پیام ۲ n بیتی
- r: بیت‌های تصادفی رمزنگاری
- خروجی شامل:
  - ciphertext
  - پیام اصلی

MACها → تابع یک‌طرفه

نویسندگان اشاره می‌کنند:

تعریف استاندارد MACها (تعریف 4.2 کتاب) نیز نشان می‌دهد که اگر MACایمن وجود داشته باشد، **تابع یک‌طرفه** نیز وجود دارد.

دلیل کلی:

• MACباید در برابر جعل حتی پس از مشاهده تعداد زیادی تگ مقاوم باشد.

•در حالت غیرمشروط (MACهای unconditional) فقط برای تعداد پیام محدود ایمن‌اند.

•بنابراین MACامن در حالت کلی → نیازمند سازه‌های محاسباتی → نیازمند تابع یک‌طرفه.

(اثبات آن پیچیده‌تر است و در این بخش ارائه نمی‌شود.)

قضیه 8.26

اگر توابع یک‌طرفه وجود داشته باشند،

آنگاه رمزنگاری متقارن Authenticated Encryptionو MAC امن نیز وجود دارند.

این یعنی:

توابع یک‌طرفه حداقل فرض لازم و کافی برای کل رمزنگاری کلید خصوصی هستند.

به عبارت دیگر:

- آن‌چه برای رمزنگاری خصوصی نیاز داریم فقط **یک تابع یک‌طرفه** است.
- نه بیشتر، نه کمتر.

این نتیجه یکی از ستون‌های نظریه رمزنگاری است و در بسیاری از مقالات بنیادی به آن استناد می‌شود.

نتیجه‌گیری بسیار مهم این بخش

قضیه 8.25

اگر توابع یک‌طرفه وجود داشته باشند→ همه موارد زیر وجود دارند:

- PRG
- PRF

8.8 – نامتمایزپذیری محاسباتی Computational Indistinguishability

مفهوم اصلی

نامتمایزپذیری محاسباتی مفهومی بنیادین در رمزنگاری است. در واقع، تقریباً تمام تعاریف امنیتی در رمزنگاری (چه در این فصل و چه در فصل ۳) مبتنی بر همین مفهوم هستند.

1

به‌طور غیررسمی:

دو توزیع X و Y **نامتمایزپذیر از نظر محاسباتی** هستند اگر هیچ الگوریتم کارای تصادفی PPT distinguisher نتواند تشخیص دهد ورودی‌اش از X بوده یا از Y، جز با احتمال ناچیز.

2

3

به بیان ساده:

اگر X و Y از دید هر الگوریتم عملی "تقریباً یکسان" دیده شوند، آنگاه می‌گوییم $X \approx Y$ نامتمایزپذیر محاسباتی‌اند.

ایده شهودی

فرض کنید یک الگوریتم Dداشته باشیم که ورودی‌اش یک رشته sاست.

اگر:

- sاز Xانتخاب شده باشد، Dمعمولاً خروجی ۱ می‌دهد.
- sاز Yانتخاب شده باشد، Dمعمولاً خروجی ۰ می‌دهد.

اگر Dبتواند این دو حالت را به‌طور قابل توجه تشخیص دهد، دو توزیع قابل تشخیص هستند.

اما اگر اختلاف احتمالات:

$$\Pr[D(X) = 1] - \Pr[D(Y) = 1]$$

بسیار کوچک و ناچیز باشد (مثلاً کوچکتر از 1/ poly(n)یا negligible)،

آنگاه Dتوان تشخیص ندارد.

این دقیقا به همان شکلی است که قبلاً PRGها را تعریف کردیم.

توزیع‌های شاخص Probability Ensembles

برای داشتن تعریف دقیق و قابل تحلیل، توزیع‌ها را به‌صورت یک **دنباله از توزیع‌ها** در نظر می‌گیریم:

$X = \{X_n\}$

$Y = \{Y_n\}$

که در هر n یک توزیع متفاوت (ولی هم مرتبط) داریم.

منظور از n معمولاً همان **پارامتر امنیت** است.

نکته: ما فقط با دسته‌ای از توزیع‌ها کار داریم که **قابل نمونه‌برداری کارا** باشند؛

یعنی الگوریتمی بتواند در زمان چندجمله‌ای، نمونه‌های X_n را بسازد.

تعریف رسمی Definition 8.29

دو توزیع Xو Yنامتمایزپذیر محاسباتی هستند، اگر:

برای **هر** تمایزدهنده کارا D،

اختلاف زیر ناچیز باشد:

$\Pr[D(1^n, X_n) = 1] - \Pr[D(1^n, Y_n) = 1] \leq \text{negligible}(n)$

نکات مهم:

- Dورودی 1^n را دریافت می‌کند تا مجاز باشد زمانش بر اساس nباشد.
- استفاده از 1^n یک استاندارد است تا طول پارامتر امنیت شناخته شود.
- negligible(n)یعنی کوچک‌تر از هر 1/ n^cبه اندازه‌ای که در عمل صفر فرض می‌شود.

به‌صورت خلاصه:

$$\left| \Pr_{x \leftarrow X_n}[D(1^n, x) = 1] - \Pr_{y \leftarrow Y_n}[D(1^n, y) = 1] \right| \leq \text{negl}(n).$$

Xو Yاز نظر محاسباتی قابل تشخیص نیستند.

ارتباط با تولیدکننده شبه‌تصادفی PRG

تعریف PRGرا حالا می‌توان بسیار دقیق‌تر بیان کرد:

یک تولیدکننده G، شبه‌تصادفی است اگر:

- خروجی‌اش بزرگ‌تر از ورودی باشد
- خروجی‌های $G(U_n)$نامتمایز از توزیع یکنواخت $U_{l(n)}$باشند

یعنی:

$$\{G(U_n)\} \approx \{U_{l(n)}\}$$

این دقیقاً همان تعریف رسمی است که می‌گوید Gیک **pseudorandom generator** است.

چند نمونه‌گیری Multiple Samples

یکی از مهم‌ترین نتایج در نامتمایزپذیری محاسباتی:

قضیه 8.31

اگر دو توزیع X و Y نامتمایزپذیر باشند،

آنگاه دریافت **تعداد چندجمله‌ای از نمونه‌ها** نیز هنوز نامتمایزپذیر است.

به زبان ساده:

در صورتی که یک نمونه از X قابل تشخیص از Y نباشد،

حتی اگر به الگوریتم تعداد زیادی نمونه از X یا Y بدهید،

باز هم نمی‌تواند تشخیص دهد.

این نتیجه، پایه بسیاری از اثبات‌های مهم است از جمله در بخش 8.5 در ساخت PRF

کاربرد مثال

اگر G یک PRG باشد و:

$G(U_n) \approx U_{2n}$

آنگاه:

$(G(U_n), G(U_n), \ldots, G(U_n))$ به تعداد $t(n)$ بار

$\approx$

$(U_{2n}, U_{2n}, \ldots, U_{2n})$

به شرط اینکه $t(n)$ چندجمله‌ای باشد.

$$\{(\underbrace{G(U_n),\ldots,G(U_n)}_{t(n)})\}_{n\in\mathbb{N}} \quad \text{and} \quad \{(\underbrace{U_{2n},\ldots,U_{2n}}_{t(n)})\}_{n\in\mathbb{N}}$$

نامتمایزپذیری محاسباتی پایهٔ تمامی موارد زیر است:

امنیت رمزنگاری متقارن

امنیت رمزنگاری نامتقارن

PRF ،PRP ،MAC ،PRG

رمزنگاری مبتنی بر شبکه فایستل

hybrid argumentاثبات‌های مبتنی بر

semantic securityامنیت

indistinguishability under chosen ciphertext attack (IND-CCA)

در حقیقت، بدون این مفهوم هیچ تعریف استاندارد امنیت در رمزنگاری امکان‌پذیر نبود.

اهمیت این مفهوم

جمع‌بندی بخش 8.8

دو توزیع زمانی نامتمایز هستند که هیچ الگوریتم کارا نتواند آن‌ها را از هم تشخیص دهد.

این مفهوم پایه‌ی اصلی تعریف PRG، PRF و تمام primitives رمزنگاری است.

استفاده از probability ensembles برای تعریف دقیق ضروری است.

تعداد چندجمله‌ای نمونه‌ها نیز امنیت را از بین نمی‌برد Theorem 8.31

تقریباً تمام اثبات‌ها در رمزنگاری مدرن مبتنی بر **Hybrid Argument** و مفهوم **Computational Indistinguishability** هستند.

با تشـــکر از توجـــه شـــما
مخصوصا شما استاد