ساختار اسفنجی؛ معرفی و کاربردها

اکرم خالصی، مجید رحیمی و محمدعلی ارومیه‌چی‌ها*

پژوهش‌گر، پژوهشگاه توسعه فناوری‌های پیشرفته خواجه نصیرالدین طوسی، پژوهشکده افتا، گروه رمز و امنیت اطلاعات
khalesiakram@yahoo.com
rahimi@rcdat.ir
Orumiehchiha@rcdat.ir

چکیده

طراحی طیف وسیعی از الگوریتم‌های رمزنگاری با استفاده از ساختار اسفنجی تنها با طراحی یک تبدیل یا جایگشت کاهش می‌یابد. درواقع با استفاده از ساختار اسفنجی می‌توان از یک تبدیل یا جایگشت با ویژگی‌های مشخص برای ساخت تابع چکیده‌ساز، الگوریتم رمز دنباله‌ای، الگوریتم رمزنگاری احراز اصالت‌شده و مولد اعداد شبه‌تصادفی استفاده کرد. امکان استفاده از یک تبدیل یا جایگشت واحد در کاربردهای مختلف از یکسو موجب ساده‌سازی طراحی گستره‌ایی از الگوریتم‌های رمزنگاری می‌شود و از سوی دیگر پیاده‌سازی را نیز ساده‌تر می‌کند. ساختار اسفنجی، مشابه یک سبک عملکرد در رمزهای قالبی، مزایایی مانند امکان ارائه امنیت اثبات‌پذیر، سادگی در طراحی الگوریتم و پیاده‌سازی آن را به‌دست می‌دهد. در این نوشتار به معرفی این ساختار و انواع آن می‌پردازیم و کاربردها و نیازهای امنیتی آن را تشریح می‌کنیم.

واژگان کلیدی: ساختار اسفنج، رمزهای دنباله‌ایی، توابع درهم‌ساز، الگوریتم رمزنگاری احراز اصالت شده، مولد اعداد شبه تصادفی

۱- مقدمه

ساختار اسفنج[1] و ساختار اسفنج دوتایی[2][۱] می‌توانند برای ساخت طیف گسترده‌ای از کاربردهای رمزنگاری شامل رمزهای دنباله‌ای، توابع چکیده‌ساز [۲، ۳، ۴]، مولد اعداد شبه‌تصادفی با امکان بارگذاری مجدد هسته [۵]، توابع تولید کلید، کدهای احراز اصالت پیام و رمزنگاری احراز اصالت‌شده [۶] مورد استفاده قرار گیرند. این امکان، استفاده از یک جایگشت واحد در کاربردهای مختلف را میسر می‌سازد؛ از این‌رو، حجم پیاده‌سازی الگوریتم‌های مورد نظر نیز کاهش می‌یابد. همچنین، طراحی و استفاده از یک جایگشت قوی بدون نگرانی در مورد اجزا (مانند طرح کلید در رمزهای قالبی) یک مزیت به‌شمار می‌رود.

ایدهٔ ساختار اسفنجی در زمان طراحی تابع چکیده‌ساز RADIOGATUN شکل گرفته است. طول ورودی و خروجی در این تابع چکیده‌ساز متغیر است. در زمان پیشنهاد این الگوریتم، طراحان می‌بایست امنیت طرح را مشخص می‌کردند. برای یک تابع چکیده‌ساز با طول چکیدهٔ ثابت n بیت، به‌طورعمومی (به‌طور ضمنی یا با صراحت) امنیت تابع به نسبت امنیت یک پیشگوی تصادفی که خروجی آن به n بیت بریده شده است، مقایسه می‌شود. بدین ترتیب، مقاومت در برابر حملات اولیه روی توابع چکیده‌ساز با طول ثابت مشخص می‌شود، مانند $2^{n/2}$ برای برخورد و 2^n برای پیش تصویر(دوم).

برای اولیه‌های رمزنگاری با طول خروجی متغیر، مانند RADIOGATUN، بیان امنیت بر اساس طول خروجی منطقی نیست؛ زیرا به‌طور ضمنی نتیجه می‌شود که با افزایش طول خروجی بدون محدودیت می‌توان امنیت را افزایش داد.

علاوه‌بر بیان امنیت در برابر حملات اولیه روی توابع چکیده‌ساز، طراحان تصمیم گرفتند امنیت قابل حصول برای یک تابع ایده‌آل را بیان کنند. برای این منظور، در [۷] یک تابع ایده‌آل معرفی کردند؛ اما پس از انتشار مقاله، متوجه ایده‌آل‌نبودن تابع شده و موضوع را عمیق‌تر مورد بررسی قرار دادند. هدف آنها تعریف تابعی با رفتاری مشابه یک پیش‌گوی تصادفی بود، تنها با این تفاوت که تابع مورد نظر شامل برخورد درونی[3] خواهد بود. نتایج این مطالعات، بسط مفهوم توابع تصادفی اسفنجی است که مشابه پیش‌گوی تصادفی هستند.

معرفی اسفنج: ساختار اسفنج یک ساختار تکرارشونده برای ساخت تابع F با طول ورودی متغیر و طول خروجی دلخواه مبتنی بر یک تبدیل یا جایگشت f با طول ثابت b بیت است (b طول نامیده می‌شود).

[1] Sponge structure
[2] Duplex construction
[3] Internal collision

*نویسندهٔ مسؤول

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

ساختار اسفنج روی حالت داخلی b-بیتی کار می‌کند که در آن $b = r + c$. مقدار r نرخ بیتی[1] و مقدار c ظرفیت[2] اسفنج نامیده می‌شود (شکل ۱).

ابتدا، تمامی بیت‌های حالت داخلی با مقدار صفر بارگذاری می‌شود. پیام ورودی، دنباله‌زنی شده و به قالب‌های r-بیتی بریده می‌شود؛ سپس دو مرحله در ساختار اسفنج اجرا می‌شود: مرحلهٔ جذب‌کردن[3] و سپس مرحلهٔ فشرده‌شدن[4].

- در مرحلهٔ جذب‌کردن، قالب‌های پیام r-بیتی با r بیت ابتدای حالت داخلی (نرخ r) XOR شده و پس از XOR هر قالب پیام، یک‌مرتبه تابع f روی حالت داخلی (b-بیتی) اسفنج اجرا می‌شود. زمانی که تمامی قالب‌های پیام پردازش (جذب) شد، اسفنج وارد مرحله فشرده‌شدن می‌شود.
- در مرحلهٔ فشرده‌شدن، r بیت ابتدایی حالت داخلی اسفنج به‌عنوان خروجی برگردانده و پس از تولید هر r بیت خروجی یک مرتبه تابع f روی حالت داخلی اسفنج اجرا و تعداد قالب‌های خروجی توسط کاربر مشخص می‌شود.

c بیت انتهایی حالت داخلی هرگز به‌صورت مستقیم از قالب‌های ورودی پیام تأثیر نگرفته و هرگز به‌صورت مستقیم به‌عنوان خروجی مورد استفاده قرار نمی‌گیرند. گفتنی است که در برخی از الگوریتم‌های طراحی‌شده با ساختار اسفنجی، ابتدا مقداری مخالف با مقدار تمام صفر در بیت‌های حالت داخلی اسفنج بارگذاری می‌شود. این مسأله تأثیری در مطالبی که در ادامه می‌آید، نخواهد داشت.

اسفنج به‌عنوان مرجع امنیتی: برای ارزیابی امنیت یک اولیه رمزنگاری، می‌توان خصوصیاتی را که بایستی در برابر آنها مقاوم باشد، به‌صورت جامع فهرست و سطح مقاومت مربوط به هر یک را مشخص کرد. روش دیگر، بررسی امنیت یک اولیه رمزنگاری[5] با توجه به یک مدل امنیتی است. بدین معنا که احتمال موفقیت حمله روی آن اولیه رمزنگاری و مدل مورد نظر با یکدیگر مقایسه شود. این روش تمام خصوصیات امنیتی، حتی آنهایی را که در فهرست یادشده پیش‌بینی نشده‌اند، نیز پوشش می‌دهد.

در توابع چکیده‌ساز با طول چکیدهٔ ثابت، امنیت مورد نیاز در برابر حملات بر اساس طول چکیده بیان می‌شود. تا چندی پیش انتظار اینکه امنیت یک تابع چکیده‌ساز در برابر حملات رایج برابر با امنیت یک پیشگوی تصادفی باشد، انتظاری منطقی بود؛ اما، این وضعیت بعد از انتشار حملات عمومی[6] تغییر کرد. با انتشار این حملات، تأثیر حافظه محدود در امنیت تابع چکیده‌ساز معرفی شد که در پیش‌گوی تصادفی مورد استفاده، به‌عنوان یک مدل امنیتی مطرح نیست.

یک تابع تکرارشونده برای ذخیره حالت داخلی و پردازش قالب به قالب ورودی از یک حافظه محدود استفاده می‌کند. در هر مقطع زمانی، حالت داخلی تابع تکرارشونده قالب‌های ورودی را که تا کنون رسیده است خلاصه می‌کند. از آنجا که تعداد بیت‌های حالت داخلی محدود است، ممکن است در حالت داخلی برخورد رخ دهد. از سوی دیگر، پیش‌گوهای تصادفی فاقد برخورد در حالت داخلی بوده و چنین مفهومی در مورد آنها مطرح نمی‌شود. این مسأله دلیل اصلی عدم امکان استفاده مستقیم از پیش‌گوهای تصادفی برای بیان امنیت توابع با طول خروجی متغیر است: تأثیر حافظه محدود که در هر اولیه رمزنگاری وجود دارد در آنها نمایش داده نمی‌شود؛ از سوی دیگر، توابع اسفنجی تصادفی می‌توانند جایگزینی برای پیش‌گوهای تصادفی در بیان امنیت باشند. یک اسفنج تصادفی نمونه‌ای از ساختار اسفنج با f انتخابی به‌صورت تصادفی از میان تبدیل‌ها (جایگشت‌ها)ی b-بیتی است. نشان داده شده است که امنیت یک تابع اسفنج تصادفی، جز در مواردی که از حافظه محدود نتیجه می‌شود، برابر با یک پیش‌گوی تصادفی است. یک اسفنج تصادفی می‌تواند به‌عنوان یک مدل مرجع برای بیان امنیت توابع چکیده‌ساز تکرارشونده و رمزهای دنباله‌ای مورد استفاده قرار گیرد.

۲- تعاریف

در بخش تعاریف، اصطلاحات و توابع کمکی در بحث ساختارهای اسفنجی را که در بخش‌های بعدی استفاده می‌شوند، معرفی می‌کنیم.

۲-۱- رشته‌بیت

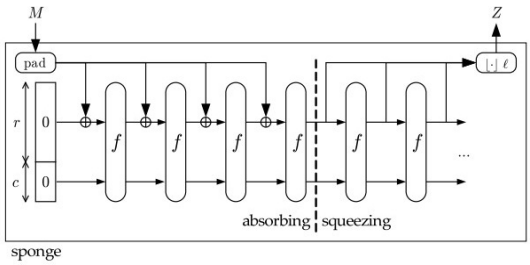
طول رشته‌بیت M بر حسب بیت را با $|M|$ نمایش می‌دهیم. رشته‌بیت M می‌تواند به‌عنوان دنباله‌ای از قالب‌ها با طول ثابت x در نظر گرفته شود که در آن طول قالب آخر ممکن است کوتاه‌تر باشد. تعداد قالب‌های M با $|M|_x$ نمایش داده می‌شود. قالب‌های M با M_i نمایش داده‌شده و اندیس آن از صفر تا $|M|_x - 1$ تغییر می‌کند. طول یک رشته تهی برابر صفر بوده و بیتی ندارد. رشتهٔ تهی را می‌توان به‌صورت

[1] Bitrate
[2] Capacity
[3] Absorbing
[4] Squeezing
[5] concrete
[6] Generic attacks

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

دنباله‌ای بدون قالب و یا یک قالب با طول صفر در نظر گرفت. در این مستند، درصورتی‌که اشاره صریح نشده باشد، رشته تهی را بدون قالب در نظر می‌گیریم.

تفکیک رشته‌بیت M به l بیت ابتدایی آن را با $\lfloor M \rfloor_l$ نمایش می‌دهیم. یک رشته‌بیت متشکل از n صفر را با 0^n و الحاق[1] دو رشته‌بیت M و N را با $M||N$ نمایش می‌دهیم.



(شکل-۱): ساختار اسفنجی $Z = SPONGE[f, pad, r](M, l)$

مجموعهٔ رشته‌بیت‌ها شامل رشته تهی و بدون رشتهٔ تهی را به‌ترتیب با Z_2^* و Z_2^+ نمایش می‌دهیم. مجموعهٔ رشته‌بیت‌ها با طول نامحدود را با Z_2^∞ نمایش می‌دهیم.

۲-۲- قوانین دنباله‌زنی

برای قوانین دنباله‌زنی[2] از نمادگذاری زیر استفاده می‌کنیم: دنباله‌زنی پیام M به یک دنباله از قالب‌های x-بیتی را با $M||pad[x](|M|)$ نمایش می‌دهیم. برای دنباله‌زنی‌های یک‌به‌یک، عنوان دنباله‌زدایی[3] را برای بازیابی M از $P = M||pad[x](|M|)$ استفاده می‌کنیم.

تعریف ۱: یک دنباله‌زنی منطبق بر اسفنج است، اگر هرگز رشته تهی نتیجه ندهد و در شرط زیر صدق کند:

$$\forall n \geq 0, \forall M, M' \in Z_2^*: M \neq M' \Rightarrow M \| pad[r](|M|) \neq M' \| pad[r](|M'|) \| 0^{nr}$$

حال دنباله‌زنی‌های ساده‌ای را که منطبق بر اسفنج هستند، معرفی می‌کنیم.

تعریف ۲: دنباله‌زنی ساده که در آن یک بیت با مقدار یک به‌همراه کمترین تعداد بیت با مقدار صفر برای آن‌که طول بیتی رشته حاصل ضریبی از طول قالب باشد، با $pad10^*$ نمایش می‌دهیم.

دنباله‌زنی ساده حداقل یک بیت و حداکثر به اندازه تعداد بیت‌های یک قالب به انتهای رشته اضافه می‌کند.

تعریف ۳: دنباله‌زنی چندنرخی[4] که در آن یک بیت با مقدار یک به‌همراه کمترین تعداد بیت با مقدار صفر به‌همراه یک بیت با مقدار یک را برای آنکه طول بیتی رشته حاصل ضریبی از طول قالب باشد با $pad101^*$ نمایش می‌دهیم.

واضح است که این دنباله‌زنی منطبق بر اسفنج و یک‌به‌یک بوده و یک رشته تهی و یا رشته‌ای با آخرین قالب تمام صفر نتیجه نمی‌دهد. دنباله‌زنی چندنرخی حداقل دو بیت و حداکثر یک واحد بیشتر از تعداد بیت‌های یک قالب به یک رشته اضافه می‌کند.

۲-۳- پیش‌گوی تصادفی، تبدیل و جایگشت

یک پیش‌گوی تصادفی را با RO نمایش داده و آن را بصورت زیر تعریف می‌کنیم:

تعریف ۴: یک پیش‌گوی تصادفی RO رشتهٔ دودویی را با طول دلخواه به‌عنوان ورودی گرفته و برای هر ورودی یک رشته تصادفی نامحدود تولید می‌کند؛ بدان معنی که پیش‌گوی تصادفی نگاشتی از Z_2^* به Z_2^∞ است.

فراخوانی RO با ورودی M را که در آن خروجی l بیت ابتدایی بریده شده است، به‌صورت $Z = RO(M, l)$ نمایش می‌دهیم.

تعریف ۵: یک تبدیل تصادفی با طول داده‌شده b، تبدیلی است که به‌صورت تصادفی و یکنواخت از میان 2^{b2^b} تبدیل b-بیتی انتخاب شده است.

تعریف ۶: یک جایگشت تصادفی با طول داده‌شده b، جایگشتی است که به‌صورت تصادفی و یکنواخت از میان $2^b!$ جایگشت b-بیتی انتخاب شده است.

۳- ساختار اسفنجی و ساختار دوتایی

در این بخش به معرفی ساختار اسفنجی و انواع آن می‌پردازیم.

۳-۱- ساختار اسفنجی

ساختار اسفنجی، تابع $SPONGE[f, pad, r]$ را با دامنه Z_2^* و برد Z_2^∞ می‌سازد که در آن f تبدیل یا جایگشت b-بیتی، pad دنباله‌زنی منطبق بر اسفنج و r پارامتر نرخ بیتی است.

ساختار اسفنجی یک حالت داخلی b-بیتی به نام s دارد. ابتدا تمام بیت‌های حالت داخلی با صفر مقداردهی می‌شوند. پیام ورودی دنباله‌زنی شده و به قالب‌های r-بیتی بریده می‌شود؛ سپس دو گام اجرا می‌شود: گام جذب‌کردن و گام فشرده‌شدن. در این گام‌ها، با r بیت ابتدایی و باقی b-r بیت از حالت داخلی s متفاوت رفتار می‌شود. r بیت ابتدایی حالت داخلی را بخش بیرونی[5] نامیده و آن را با $\bar{s}$ نمایش می‌دهیم. همچنین، b-r

[1] Concatenation
[2] Padding
[3] Unpadding
[4] Multi-rate Padding
[5] Outer part

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

بیت باقی از حالت داخلی s را بخش درونی[1] نامیده و آن را با $\hat{s}$ نمایش می‌دهیم. طول بخش درونی[2] b-r بیت بوده و ظرفیت c نامیده می‌شود. دو گام یادشده به‌شرح زیر است:

گام جذب‌کردن: قالب‌های r-بیتی ورودی با بخش بیرونی حالت داخلی XOR شده و پس از جذب هر قالب r-بیتی تابع f اجرا می‌شود. زمانی‌که تمام قالب‌های پیام پردازش(جذب) شد، ساختار اسفنجی وارد گام فشرده‌شدن می‌شود.

گام فشرده‌شدن: بخش بیرونی حالت داخلی به‌عنوان خروجی برگردانده شده و پس از تولید هر r بیت خروجی، تابع f روی حالت داخلی اسفنج اجرا می‌شود. تعداد دفعات تکرار با توجه به طول خروجی درخواستی l تعیین و درنهایت خروجی اسفنج به‌عنوان خروجی معرفی می‌شود. c بیت حالت داخلی هرگز به‌طور مستقیم از قالب‌های ورودی تأثیر نگرفته و هرگز در گام فشرده‌شدن به‌طور مستقیم به‌عنوان خروجی معرفی نمی‌شود. اثبات می‌شود که ظرفیت c تعیین‌کنندهٔ میزان امنیت قابل حصول ساختار است.

از اصطلاح اسفنج تصادفی برای اسفنجی با یک تبدیل یا جایگشت تصادفی f استفاده می‌کنیم؛ همچنین منظور از حمله عمومی روی ساختار اسفنجی در این مستند به‌شرح زیر است:

تعریف ۷: اگر یک حمله روی یک تابع اسفنجی از خصوصیات f استفاده نکند حمله‌ای عمومی خواهد بود.

ساختار اسفنجی در شکل (۱) نشان داده شده و الگوریتم (۱) در شکل (۲) تعریفی برای آن ارائه می‌کند.

Algorithm 1 The sponge construction $\text{SPONGE}[f, \text{pad}, r]$
Require: $r < b$

Interface: $Z = \text{sponge}(M, \ell)$ with $M \in \mathbb{Z}_2^*$, integer $\ell > 0$ and $Z \in \mathbb{Z}_2^\ell$
$P = M||\text{pad}[r](|M|)$
$s = 0^b$
for $i = 0$ to $|P|_r - 1$ **do**
  $s = s \oplus (P_i||0^{b-r})$
  $s = f(s)$
end for
$Z = \lfloor s \rfloor_r$
while $|Z|_r r < \ell$ **do**
  $s = f(s)$
  $Z = Z||\lfloor s \rfloor_r$
end while
return $\lfloor Z \rfloor_\ell$

(شکل-۲): الگوریتم ساختار اسفنجی [۱]

۳-۲- ساختار دوتایی

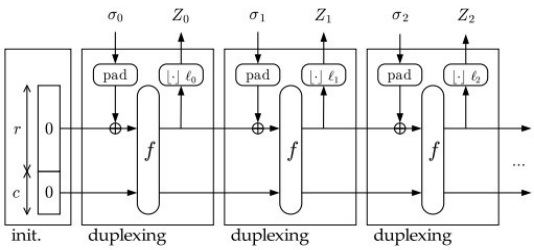
مشابه ساختار اسفنجی، ساختار دوتایی[3] $DUPLEX[f, pad, r]$ از یک تبدیل یا جایگشت با طول ثابت f، یک قاعده دنباله‌زنی و پارامتر نرخ بیتی r برای ساخت یک طرح رمزنگاری استفاده می‌کند. ساختار دوتایی یک شیء[4] را نتیجه می‌دهد که یک رشتهٔ ورودی را گرفته و خروجی را برمی‌گرداند؛ با این ویژگی که خروجی یادشده به تمام ورودی‌هایی که تا کنون اعمال شده وابسته است. نمونه‌ای برای یک ساختار دوتایی را یک شیء دوتایی[5] نامیده و آن را با D نمایش می‌دهیم. فراخوانی یک شیء دوتایی مشخص D را با پیشوند D و یک نقطه نمایش می‌دهیم.

یک شیء دوتایی D یک حالت داخلی b-بیتی دارد. در ابتدای مرحله مقداردهی اولیه حالت داخلی با صفر مقداردهی شده و پس از اتمام مقداردهی اولیه فراخوانی به‌صورت $D.duplexing(\sigma, l)$ که σ رشتهٔ ورودی و l تعداد بیت‌های درخواستی خروجی است، انجام می‌شود. بیشترین تعداد بیت قابل درخواست l برابر با r بوده و طول رشته ورودی σ بایستی حداکثر به اندازه‌ای باشد که بعد از دنباله‌زنی تنها یک قالب r-بیتی نتیجه دهد. بزرگ‌ترین طول σ را نرخ دوتایی بیشینه[6] نامیده و آن را با $\rho_{max}(pad, r)$ نمایش می‌دهیم.

$$\rho_{max}(pad, r) = min\{x: x + |pad[r](x)| > r\} - 1$$

واضح است که نرخ دوتایی بیشینه از نرخ بیتی کوچک‌تر بوده و با انتخاب دنباله‌زنی‌ای که کمترین تعداد بیت ممکن را به رشته ورودی اضافه می‌کند، بیشینه می‌شود.

با دریافت یک فراخوانی $D.duplexing(\sigma, l)$، شیء دوتایی رشتهٔ ورودی σ را دنباله‌زنی کرده و آن را با بخش بیرونی حالت داخلی XOR می‌کند؛ سپس f روی حالت داخلی اسفنج اعمال شده و l بیت ابتدایی از بخش بیرونی حالت داخلی به‌عنوان خروجی معرفی می‌شود. فراخوانی با ورودی رشته تهی σ را فراخوانی خالی[7] و فراخوانی با l=0 را فراخوانی صامت[8] می‌نامیم. ساختار دوتایی در شکل (۳) نشان داده شده و الگوریتم (۲) (شکل ۴) تعریفی برای آن ارائه می‌کند.



(شکل-۳): ساختار دوتایی

[1] Inner part
[2] Inner state
[3] Duplex construction
[4] Object
[5] Duplex object
[6] Maximum duplex rate
[7] Blank call
[8] Mute call

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

خروجی فراخوانی دوتایی خروجی تابع اسفنجی بوده و ثابت می‌شود که امنیت ساختار دوتایی برابر با امنیت ساختار اسفنجی است.

Algorithm 2 The duplex construction $\text{DUPLEX}[f, \text{pad}, r]$
Require: $r < b$
Require: $\rho_{\max}(\text{pad}, r) > 0$

Interface: $D.\text{initialize}()$
$s = 0^b$

Interface: $Z = D.\text{duplexing}(\sigma, \ell)$ with $\ell \le r, \sigma \in \bigcup_{n=0}^{\rho_{\max}(\text{pad},r)} \mathbb{Z}_2^n$, and $Z \in \mathbb{Z}_2^\ell$
$P = \sigma || \text{pad}[r](|\sigma|)$
$s = s \oplus (P || 0^{b-r})$
$s = f(s)$
return $\lfloor s \rfloor_\ell$

(شکل-۴): الگوریتم ساختار دوتایی [۱]

۴- امنیت

تا کنون مقالات زیادی در زمینه ارزیابی امنیت اسفنج به چاپ رسیده است. برخی از این مقالات به ارائه امنیت اثبات‌پذیر پرداخته [۸، ۹ و ۱۰] و برخی دیگر نیز به ارزیابی امنیت انواع الگوریتم‌های مبتنی بر این ساختار پرداخته‌اند [۱۱، ۱۲]. در این بخش ابتدا توابع کمکی مورد استفاده در ارزیابی امنیت تابع اسفنجی معرفی و سپس حملات اولیه شامل برخورد، بازیابی حالت داخلی و انقیاد خروجی[1] معرفی می‌شود.

۴-۱- توابع کمکی

در اینجا دو تابع کمکی[2] تعریف می‌کنیم که تعاریف و استدلال‌ها را در بررسی حملات روی ساختار اسفنجی ساده می‌کنند.

۴-۱-۱- تابع جذب و مسیر

تابع کمکی نخست تابع جذب $s = ABSORB[f, r](P)$ است. این تابع رشته P که $|P|$ ضریبی از r است را به‌عنوان ورودی گرفته و مقدار حالت داخلی را بعد از جذب P برمی‌گرداند. تابع جذب در الگوریتم (۳) (شکل ۵) تعریف شده است. از $s = absorb(P)$ برای نمایش تابع جذب با جایگشت با تبدیل f مشخص و پارامتر نرخ بیتی r مشخص استفاده می‌کنیم.

تعریف ۸: P را مسیری به وضعیت s می‌گوییم اگر $s = absorb(P)$ باشد.

قالب j-ام خروجی از یک تابع اسفنجی متناظر با ورودی M برابر است با:

$Z_j = \overline{absorb}(P||0^{rj}), j \ge 0$

که در آن $P = M||pad[r](|M|)$ است.

تابع جذب برای بیان تغییر حالت داخلی در هر دو صورت جذب ورودی M و فشرده‌شدن قابل استفاده است.

Algorithm 3 The absorbing function $\text{ABSORB}[f, r]$
Require: $r < b$

Interface: $s = \text{absorb}(P)$ with $P \in \mathbb{Z}_2^*$, and $s \in \mathbb{Z}_2^b$
$s = 0^b$
for $i = 0$ to $|P|_r - 1$ **do**
  $s = s \oplus (P_i || 0^{b-r})$
  $s = f(s)$
end for
return s

(شکل-۵): الگوریتم تابع جذب [۱]

۴-۱-۲- تابع فشردن

تابع کمکی دوم که به جهت ساختاری مشابه تابع جذب‌کردن است، تابع فشردن $Z = SQUEEZE[f, r](s, l)$ است. از $Z = squeeze(s, l)$ برای نمایش تابع فشردن با جایگشت با تبدیل f مشخص و پارامتر نرخ بیتی r مشخص استفاده می‌کنیم.

برای یک حالت داخلی داده شده s، $squeeze(s, l)$ معرف خروجی بریده به طول l بیت از تابع اسفنجی، با حالت داخلی s در ابتدای گام فشردن است. تعریف تابع فشردن در الگوریتم (۴) (شکل ۶) آورده شده است.

Algorithm 4 The squeezing function $\text{SQUEEZE}[f, r]$
Require: $r < b$

Interface: $Z = \text{squeeze}(s, \ell)$ with $s \in \mathbb{Z}_2^b$, integer $\ell > 0$ and $Z \in \mathbb{Z}_2^\ell$
$Z = \lfloor s \rfloor_r$
while $|Z|_r r < \ell$ **do**
  $s = f(s)$
  $Z = Z || \lfloor s \rfloor_r$
end while
return $\lfloor Z \rfloor_\ell$

(شکل-۶): الگوریتم تابع فشردن [۱]

۴-۲- حملات اولیه روی تابع اسفنجی

در این بخش تعدادی از حملات روی توابع اسفنجی را بررسی می‌کنیم. گفتنی است که این حملات ناشی از حالت داخلی محدود توابع اسفنجی بوده، لذا در مورد پیش‌گوی تصادفی مصداق ندارد. حملات مورد نظر باند بالایی برای امنیت قابل ارائه توسط یک تابع اسفنجی قرار می‌دهند و حملات پایه‌ای هستند. به این دلیل این حملات را حملات اولیه[3] می‌نامیم.

[1] Output binding
[2] Auxiliary function
[3] Primary attacks

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

یک ساختار اسفنجی به‌صورت زنجیره‌ای از توابع دنباله‌زنی، جذب و فشردن است. برای $Z = SPONGE[f,pad,r](M,l)$ داریم:

$P = M||pad[r](|M|)$
$s = ABSORB[f,r](P)$
$Z = SQUEEZE[f,r](s,l)$

وارون‌سازی توابع کمکی ساختار اسفنجی پیچیده بوده و در حالت کلی، یافتن یک مسیر P به یک حالت داخلی داده شده s و نیز یافتن حالت داخلی s برای یک خروجی داده شده Z دشوار است. علاوه‌براین، یافتن دو مسیر متفاوت به یک حالت داخلی (برخورد حالت داخلی[1]) نیز پیچیده است.

۴-۲-۱- برخورد

تعریف ۹: برای زوجی از مسیرهای متفاوت $(P \neq Q)$ برخورد حالت داخلی عبارت است از:

$absorb(P) = absorb(Q)$

برخورد حالت داخلی را از نظر گام اجرایی تابع اسفنجی در زمان وقوع برخورد (جذب‌کردن و یا فشرده‌شدن) و نیز بیت‌هایی که برخورد در آنها رخ می‌دهد (بیت‌های بخش درونی و یا بخش بیرونی) می‌توان بررسی کرد.

بسته به اینکه برخورد حالت داخلی در چه گامی رخ دهد، نتایج مختلف محدودبودن حالت داخلی را نشان می‌دهد. برخورد در گام جذب‌کردن می‌تواند خروجی‌های یکسان نتیجه دهد؛ به‌عنوان مثال اگر $absorb(P) = absorb(Q)$، با فشرده‌شدن خروجی یکسانی $absorb(P||0^{rj}) = absorb(Q||0^{rj})$ برای هر مقدار j نتیجه می‌شود. برخورد در گام فشرده‌شدن می‌تواند بیان‌گر دوره در دنباله خروجی باشد؛ به‌عنوان نمونه، اگر به‌ازای مقداری برای P و d داشته باشیم $absorb(P) = absorb(P||0^{dr})$، دنباله خروجی متناوب خواهد بود.

در بررسی موقعیت بیت‌هایی که برخورد در آنها اتفاق می‌افتد داریم:

تعریف ۱۰: یک برخورد داخلی[2] زوجی از مسیرهای متفاوت $P \neq Q$ به حالت داخلی یکسان است: $\widehat{absorb}(P) = \widehat{absorb}(Q)$.

واضح است که برخورد حالت داخلی بیان‌گر برخورد داخلی نیز هست؛ اما وارون این مسأله برقرار نیست. گفتنی است، ایجاد برخورد حالت داخلی از یک برخورد داخلی ساده است. برای $P \neq Q$ داده‌شده به‌گونه‌ای که $\widehat{absorb}(P) = \widehat{absorb}(Q)$، می‌توان برخورد حالت داخلی با $P||A \neq Q||B$ برای هر $A,B \in Z_2^r$ با شرط $\overline{absorb}(P) \oplus A = \overline{absorb}(Q) \oplus B$ را یافت.

[1] State collision
[2] Inner collision

۴-۲-۲- انقیاد خروجی و بازیابی حالت داخلی

در حالت کلی، یافتن حالت داخلی s به‌گونه‌ای که $squeeze(s,|Z|) = Z$، برای رشته‌های طولانی Z دشوار است. بسته به دنباله مورد بررسی Z و هدف مهاجم، دو حالت داخلی را در نظر می‌گیریم: انقیاد خروجی و بازیابی حالت داخلی.

در انقیاد خروجی رشته مورد بررسی Z به‌الزام نتیجه فشرده‌شدن یک حالت داخلی نبوده و بنابراین ممکن است، درعمل پاسخی وجود نداشته باشد.

تعریف ۱۱: برای یک رشته داده شده Z، انقیاد خروجی یافتن حالت داخلی s است به‌گونه ای که $squeeze(s,|Z|) = Z$.

تعداد حالت‌های داخلی که با فشرده‌شدن، رشتهٔ داده‌شدهٔ Z را نتیجه می‌دهند، برابر است با $2^{b-|Z|}$. اگر $|Z| > b$ باشد، احتمال وجود چنین وضعیتی به‌طورتقریبی برابر است با $2^{b-|Z|}$.

در بازیابی حالت داخلی برخلاف انقیاد خروجی، رشته Z با فشرده‌شدن یک حالت داخلی به‌دست آمده است. ممکن است، حالت‌های داخلی دیگری $s' \neq s$ باشد که با فشرده‌شدن، رشته Z را نتیجه دهد، اما حالت داخلی s به‌عنوان تنها پاسخ در نظر گرفته می‌شود.

تعریف ۱۲: بازیابی حالت داخلی شامل یافتن حالت داخلی s برای رشته Z بصورت $Z = squeeze(s,|Z|)$ است.

اگر $|Z| > b$ باشد، با احتمال بالایی تنها یک پاسخ وجود داشته و انقیاد خروجی بازیابی حالت داخلی s که رشته مورد نظر از آن تولیدشده را نتیجه می‌دهد. اگر $|Z| < b$ باشد، به‌طورعمومی چند حالت داخلی وجود دارد که فشرده‌شدن آنها رشته Z را نتیجه می‌دهد و انقیاد خروجی به‌الزام بازیابی حالت داخلی را نتیجه نمی‌دهد.

۵- سبک عملکرد در اسفنج

تابع اسفنجی تنها یک ورودی M که رشته‌ای با طول دلخواه است را دریافت می‌کند. برخلاف برخی ساختارها، ساختار اسفنجی ورودی دیگری به‌عنوان بردار اولیه ندارد. سبک‌های عملکرد مختلف توابع اسفنجی شامل روش‌های نگاشت انواع ورودی‌ها همچون کلید، بردار اولیه، پیام و ... به ورودی اسفنج (M) و برش خروجی به طول دلخواه است. در این بخش روش‌های پایه را که برای ساخت سبک عملکرد قابل استفاده هستند، معرفی می‌کنیم.

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

۵-۱- تفکیک دامنه

به‌دلیل ویژگی‌های تصادفی‌بودن و ورودی‌های با طول دلخواه، یک پیش‌گوی تصادفی می‌تواند با استفاده از سازوکار تفکیک دامنه[1] برای پیاده‌سازی چند پیش‌گوی تصادفی مورد استفاده قرار گیرد. برای این منظور کافی است دامنه را به دسته‌هایی تفکیک کنیم. تفکیک رشته‌ها به دو دسته شامل رشته‌هایی که با صفر شروع می‌شوند و رشته‌هایی که با یک شروع می‌شوند، نمونه‌ای از تفکیک دامنه است. با یک پیش‌گوی تصادفی مشخص RO می‌توان دو پیش‌گوی تصادفی مستقل $RO_0(M) \triangleq RO(0||M)$ و $RO_1(M) \triangleq RO(1||M)$ را تعریف کرد. این یک ابزار قدرتمند برای ساخت توابع متنوع با استفاده از یک پیش‌گوی تصادفی است. سازوکار تفکیک دامنه روی توابع اسفنجی نیز قابل اجرا است.

۵-۲- استفاده از کلید

با گنجاندن کلید در ورودی یک تابع اسفنجی می‌توان آن را به یک تابع کلیددار تبدیل کرد. در ساده‌ترین شکل، ورودی M شامل پیوست کلید K و ورودی M' به‌صورت $M = K||M'$ یا $M = M'||K$ است. چنین تابعی شبه‌تصادفی بوده و آن را با $F_K(M')(PRF)$ نمایش می‌دهیم. اگر تابع اسفنجی مشابه یک پیش‌گوی تصادفی رفتار کند، PRF برای کسی که به کلید دسترسی نداشته و تنها به تابع اسفنج دسترسی دارد مشابه یک تابع تصادفی عمل می‌کند. کلید را می‌توان قبل یا بعد از پیام قرار داد. قراردادن کلید قبل از پیام امکان پیش‌محاسبات حالت داخلی[2] را فراهم کرده و مقاومت بیشتری در برابر حملات عمومی خواهد داشت.

۵-۳- پیش‌محاسبات حالت داخلی

یک تابع اسفنجی ورودی M را به‌صورت قالب‌های r-بیتی پردازش می‌کند. در مرحله پردازش و جذب ورودی می‌توان از نوعی دنباله‌زنی استفاده کرد. برای مثال، اگر در یک اسفنج کلیددار، کلید K به‌صورت یک قالب ورودی کامل دنباله‌زنی شود، می‌توان حالت داخلی حاصل پس از جذب کلید را محاسبه و ذخیره کرد. بدین ترتیب، در زمان اجرای اسفنج با این کلید به‌خصوص، می‌توان به‌طورمستقیم محاسبات را از مقدار ذخیره‌شده برای حالت داخلی اسفنج شروع کرده و از فراخوانی یک تابع f اضافی جلوگیری کرد. از این نظر، بهتر است پارامترهای ورودی که کمترین میزان تغییرات را دارند، در ابتدا قرار دهیم. این روش برای سبک‌های عملکردی است که در آن برخی از ورودی‌ها در بسیاری از اجراها تغییر نمی‌کنند، مانند کلید یا اسم یک فضای اسم.

۶- کاربردهای اسفنج

در این بخش به معرفی نمونه‌هایی برای سبک‌های عملکرد ساختار اسفنج شامل تابع چکیده‌ساز [۲، ۳ و ۴]، رمزنگاری احراز اصالت‌شده [۶] و مولد اعداد شبه‌تصادفی [۵] که طیف وسیعی از توابع رمزنگاری را نتیجه می‌دهند، می‌پردازیم.

۶-۱- تابع چکیده‌ساز

چکیده‌ساز Keccak [۲]، یا همان الگوریتم استاندارد SHA3، از نمونه‌های شناخته‌شده برای استفاده از ساختار اسفنجی در ساخت الگوریتم‌های رمزنگاری است. با برش خروجی یک تابع اسفنجی می‌توان از آن به‌صورت یک تابع چکیده‌ساز n-بیتی استفاده کرد. اگر بخواهیم از تابع چکیده‌ساز به‌صورت چکیده‌ساز تصادفی استفاده کنیم، یک مقدار تصادفی (برای مثال salt) به پیام افزوده می‌شود. در صورت نیاز به چند تابع چکیده‌ساز می‌توان از ایده تفکیک دامنه استفاده کرد.

یک تابع چکیده‌ساز را می‌توان به‌صورت فرآیند محاسبه چکیده پیام‌ها با فراخوانی یک تابع تعریف کرد. تابع یادشده می‌تواند یک تابع فشرده‌ساز با طول ورودی ثابت، یک جایگشت و یا یک تابع چکیده‌ساز باشد. دو هدف برای یک سبک عملکرد چکیده‌ساز وجود دارد. هدف نخست ساخت یک تابع چکیده‌ساز با طول ورودی متغیر با استفاده از یک تابع با طول ورودی ثابت و هدف دوم ساخت یک تابع چکیده‌ساز درختی[3] است.

در چکیده‌ساز درختی امکان پردازش هم‌زمان بخش‌های مختلف پیام وجود دارد. ازاین‌رو، چکیده‌ساز درختی می‌تواند برای سرعت‌بخشیدن به محاسبات یک تابع چکیده‌ساز با استفاده از موازی‌سازی در معماری‌های مدرن استفاده شود. در ادامه به معرفی این ساختار و اصطلاحات مورد نیاز آن می‌پردازیم.

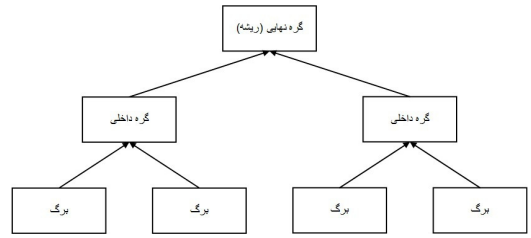
یک درخت را در نظر بگیرید که گره‌های آن را با Z_α نمایش داده و α اندیس گره نامیده می‌شود. اصطلاحات مورد استفاده در بحث چکیده‌ساز درختی به‌صورت زیر است:

- یک گره می‌تواند یک گره والد[4] واحد داشته باشد. اندیس گره والد گره‌ای با اندیس α به‌صورت $parent(\alpha)$ است.

[1] Domain separation
[2] State precomputation
[3] Tree hash function
[4] Parent node

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

در درخت، تمام گره‌ها به‌جز یک گره، یک والد دارند. گره بدون والد را گره نهایی نامیده و آن را با * نمایش می‌دهیم.
- سایر گره‌ها گره‌های داخلی[1] نامیده می‌شود (شکل ۷).

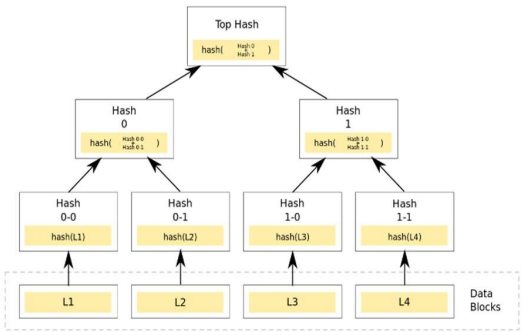


(شکل-۷): ساختار درخت

- گره با اندیس α فرزند گره با اندیس $parent(\alpha)$ نامیده می‌شود. هر گره می‌تواند صفر، یک و یا بیشتر فرزند داشته باشد. تعداد فرزندان یک گره را درجه[2] آن گره نامیده و گره بدون فرزند را برگ[3] می‌نامیم.
- گره Z_α را نیا[4] گره Z_β می‌نامیم اگر $\alpha = parent(\beta)$ و یا Z_α نیای والد گره Z_β باشد. به‌عبارت دیگر، Z_α را نیای گره Z_β می‌گوییم اگر دنباله‌ای از اندیس‌ها $\alpha_0, \alpha_1, \dots, \alpha_{d-1}$ وجود داشته باشد به‌گونه‌ای که $\alpha = \alpha_0$ ، $\alpha_{i-1} = parent(\alpha_i)$ و Z_β . $\alpha_{d-1} = parent(\beta)$ را نوه[5] گره Z_α نامیده و d فاصله بین Z_α و Z_β است. واضح است که گره نهایی بدون نیا بوده و یک گره برگ فاقد نوه است.
- هر گره Z_α از نوادگان گره نهایی بوده و فاصله بین هر گره از گره نهایی ارتفاع[6] آن نامیده می‌شود. بنابراین، ارتفاع گره نهایی صفر بوده و ارتفاع درخت برابر با بیشترین ارتفاع گره‌های آن است.

در یک نگاه، این ساختار به‌صورت زیر کار می‌کند. یک درخت ریشه‌دار با گره‌های داخلی و برگ‌ها را در نظر بگیرید. گره نهایی ریشهٔ درخت نامیده می‌شود. پیام ورودی به قالب‌هایی بریده شده و در میان برگ‌ها پخش می‌شود. روی هر برگ F اعمال شده و C بیت از خروجی برای تولید زنجیره بریده می‌شود. یک گره داخلی مقادیر زنجیره را از فرزندانش جمع‌آوری کرده، آنها را به هم پیوست کرده و با اعمال مجدد F مقدار زنجیره را محاسبه می‌کند. این عملیات به‌صورت بازگشتی تا رسیدن به گره نهایی تکرار می‌شود. خروجی تابع چکیده‌ساز درختی با اعمال F روی گره نهایی که به یک خروجی با طول نامعلوم منتج می‌شود، به‌دست می‌آید (شکل ۸). فراخوانی F توسط گره‌های مختلف، داده‌های مختلف را پردازش کرده و قابلیت موازی‌سازی دارد.

از آنجا که طول پیام ورودی، دلخواه و از پیش نامعلوم است، لازم است تا نحوه رشد درخت و یا چگونگی پذیرش قالب‌های ورودی (که تعداد آنها افزایش می‌یابد) توسط درختی با تعداد گره ثابت تعیین شود.



(شکل-۸): چکیده‌ساز درختی

۶-۱-۱- خصوصیات

سبک عملکرد چکیده‌ساز درختی از دو حالت داخلی پشتیبانی می‌کند:

رشد گره نهایی: درجه گره نهایی به‌صورت تابعی از طول پیام ورودی رشد کرده و تعداد برگ‌ها به نسبت زیاد می‌شود.

جانمایی برگ‌ها: اندازه درخت و تعداد برگ‌ها با پارامترهای سبک عملکرد درخت و مستقل از طول پیام تعیین می‌شود؛ اما قالب‌های پیام میان برگ‌ها جانمایی می‌شوند.

سبک عملکرد چکیده‌ساز درختی دو ورودی دریافت می‌کند: یک رشته دودویی پیام M و مجموعه‌ای از پارامترها که با A نمایش داده می‌شود:
- سبک عملکرد رشد درخت شامل رشد گره نهایی و یا جانمایی برگ‌ها
- ارتفاع درخت H، H>0
- درجه D گره‌ها
- اندازه قالب برگ‌ها B
- اندازه مقدار زنجیره‌ای C.

زمانی‌که رشد درخت از نوع جانمایی برگ‌هاست، درخت یک درخت ریشه‌دار با ارتفاع H است: درجه تمام گره‌های داخلی برابر D است. زمانی‌که رشد درخت از نوع رشد گره نهایی است، درجه گره نهایی به طول پیام بستگی داشته و درجه سایر گره‌ها برابر D است.

[1] Inner nodes
[2] Degree
[3] Leaf
[4] Ancestor
[5] Descendant
[6] Height

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

درخت H+1 سطح از گره‌ها دارد که با k اندیس‌گذاری می‌شود. در سطح صفر، گره‌های برگ را داریم که حاوی بیت‌های پیام هستند. گره‌ها در سطح‌های دیگر k>1 حاوی نتیجه پیوست مقادیر زنجیره‌ای هستند که هر یک از آنها نتیجه اعمال F روی گره‌ها در سطح k-1 هستند که C بیت از خروجی آنها بریده شده است. در سطح H تنها یک گره که همان گره نهایی است وجود دارد. خروجی نهایی حاصل اعمال F روی این گره است؛ حال می‌توان با تشریح نحوه شکل‌گیری گره‌ها به معرفی این سبک عملکرد پرداخت. تمامی گره‌ها به دو نوع فریم بیت ختم می‌شوند. بیت نخست تعیین می‌کند که گره یک برگ است (۱) یا خیر (۰) و بیت دوم تعیین می‌کند که گره یادشده گره نهایی است (۱) یا خیر(۰). در ادامه نحوه شکل‌گیری باقی بیت‌های هر گره را تشریح می‌کنیم.

تعداد قالب‌های B-بیتی در پیام را با $|M|_B$ نمایش داده و قالب‌های پیام را از صفر تا $|M|_B - 1$ اندیس‌گذاری می‌کنیم. توجه کنید که ممکن است، قالب آخر کمتر از B بیت داشته باشد. تعداد گره‌های برگ را با L نمایش می‌دهیم. اندیس‌گذاری گره‌ها در هر سطح را از صفر شروع می‌کنیم. گره با اندیس i در سطح k با $0 < k < H$، شامل دنباله‌ای از D مقدار زنجیره‌ای متناظر با گره‌ها در سطح k-1 با اندیس‌های iD تا $i(D+1)-1$ است.

اگر سبک عملکرد جانمایی برگ‌ها باشد، داریم $L = D^H$ و در سطح k تعداد گره‌ها برابر با D^{H-k} است. برگ با اندیس i حاوی دنباله قالب‌های پیام $M_i, M_{i+L}, M_{i+2L}, \dots$ است. گره نهایی مقادیر دنباله‌ای D گره در سطح II-1 را دارد و به‌دنبال آن دو بایت حاوی پارامترهای درخت H، D، B و C کدگذاری شده و یک بایت حاوی سبک عملکرد جانمایی برگ‌ها قرار داده می‌شود.

اگر سبک عملکرد رشد گره نهایی باشد، داریم $L = RD^{H-1}$ با $R = \left\lceil \frac{|M|_B}{D^{H-1}} \right\rceil$ و $D^{H-(k+1)}$ گره در سطح k وجود دارد. اگر $i < |M|_B$، برگ با اندیس i حاوی قالب پیام i-ام است و در غیر این‌صورت بدون قالب پیام است. گره نهایی مقادیر دنباله‌ای R گره در سطح H-1 را دارد و بدنبال آن دو بایت حاوی پارامترهای درخت H، D، B و C کدگذاری شده و یک بایت حاوی سبک عملکرد رشد گره نهایی قرار داده می‌شود.

اگر تعداد بهینه پردازش‌ها معلوم باشد، به‌سادگی می‌توان از سبک عملکرد جانمایی برگ‌ها با H=1 و D بزرگ‌تر یا مساوی با تعداد پردازش‌ها استفاده کرد؛ در این‌صورت، چکیده‌ساز درختی به یک چکیده‌ساز موازی ساده کاهش می‌یابد که در آن قالب‌های B-بیتی از پیام ورودی به D تابع اسفنجی مختلف ارسال می‌شوند. D نتیجه حاصل نیز در گره نهایی با یکدیگر ترکیب شده و خروجی نهایی تولید می‌شود.

علاوه بر سبک‌های عملکرد رشد گره نهایی و جانمایی برگ‌ها، می‌توان درخت را با افزایش ارتفاع (H)، تا جایی که تعداد برگ‌ها (L) به نسبت طول پیام باشد، بزرگ کرد. تنظیم سبک عملکرد جانمایی برگ‌ها در این شرایط قالب‌های پیام را جانمایی نمی‌کند، اما درخت را ثابت می‌کند.

سبک عملکرد چکیده‌ساز درختی دارای ویژگی درستی[1] است. ویژگی درستی برای یک تابع چکیده‌ساز بیان‌گر آن است که برتری تمایز یادشده از یک تابع چکیده‌ساز ایده‌آل دارای باند بالایی برابر با $\frac{q^2}{2^{n+1}}$ است، که در آن q تعداد پرسمان‌ها از تابع چکیده‌ساز و n طول مقادیر زنجیره‌ای است. در [۱] برای درست‌بودن یک تابع چکیده‌ساز چهار شرط بیان شده که سبک عملکرد مورد نظر شرایط یادشده را برآورده کرده و لذا دارای ویژگی درستی است.

۶-۲- رمزنگاری احراز اصالت‌شده

رمزنگاری احراز اصالت‌شده (AE) در دهه گذشته مورد توجه جامعه رمزنگاری بوده است. برگزاری مسابقه CAESAR در این زمینه گواهی بر این ادعاست [۱۴]. استفاده از رمزهای قالبی در سبک‌های عملکردی که تمامیت و محرمانگی داده را به همراه دارند، روشی برای دست‌یابی به این هدف است. تا کنون سبک‌های عملکرد زیادی برای این منظور ارائه شده‌اند که بسیاری از آنها از امنیت اثبات‌پذیر در برابر حملات عمومی نیز برخوردار هستند [۱۵، ۱۶، ۱۷، ۱۸، ۱۹]. استفاده از ساختار اسفنجی در طرح یک الگوریتم رمزنگاری احراز اصالت شده مشابه استفاده از رمزهای قالبی در سبک عملکرد است، با این تفاوت که بجای طراحی یک رمز قالبی، تنها به طراحی یک جایگشت نیاز است.

رمزنگاری احراز اصالت شده برای انتقال کلید محرمانه با حفظ تمامیت آن قابل استفاده است. این عمل که پوشش کلید[2] نامیده می‌شود، در بحث مدیریت کلید از اهمیت بالایی برخوردار است. پوشش کلید، درصورتی‌که یک شناسه واحد به هر کلید اختصاص داده شود، با استفاده از ساختار اسفنجی نیز قابل انجام است.

[1] Soundness
[2] Key wrapping

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

۶-۲-۱- مدل‌کردن رمزنگاری احراز اصالت شده

رمزنگاری احراز اصالت شده به‌صورت فرآیندی که کلید K، سرآیند داده A و بدنه داده B را دریافت کرده و متن رمزشده C و برچسب T را برمی‌گرداند، در نظر گرفته می‌شود. این عملیات را پوشش[1] نامیده و عمل دریافت سرآیند داده A، متن رمزشده C و برچسب T و بازگرداندن بدنه داده B را درصورتی‌که برچسب T صحیح باشد، واپیچیدن[2] می‌نامیم.

متن رمزشده، حاصل رمزگذاری بدنه داده B با کلید K بوده و برچسب یادشده یک MAC محاسبه شده از سرآیند و بدنه داده با کلید K است.

در اینجا فرض بر آن است که عملیات پوشش و واپیچی قطعی[3] باشد؛ بنابراین، دو ورودی یکسان (A,B) و (A',B')، با کلید یکسان K، همواره خروجی یکسان (C,T) را نتیجه می‌دهند. درصورتی‌که این ویژگی مشکل‌ساز باشد، می‌توان با استفاده از تک‌شمار[4] بر آن چیره شد.

۶-۲-۲- نیازهای امنیتی

طرح رمزنگاری احراز اصالت شده که به‌ازای کلید انتخاب‌شده به‌صورت تصادفی K، نیازهای امنیتی زیر را برآورده کند، بسیار مفید خواهد بود:

عملی‌نبودن بازیابی کلید: احتمال موفقیت یافتن کلید در حمله‌ای شامل بررسی N مقدار برای کلید بیشتر از $N2^{-|K|}$ نباشد.

عملی‌نبودن جعل برچسب: بدون بازیابی کلید، احتمال موفقیت جعل برچسب برای هر (A,B) منتخب برابر با 2^{-T} است. این احتمال برای مهاجمی که به متن رمزشده C و خروجی (C_i,T_i) متناظر با متن انتخابی (A_i,B_i) تنها با شرط $(A_i,B_i)\neq(A,B)$ دسترسی دارد نیز یکسان است.

عملی‌نبودن بازیابی متن اصلی: برای خروجی (C,T) داده شده متناظر با (A,B)، به‌ازای A منتخب و B نامعلوم، حتی برای مهاجمی که به خروجی (C_i,T_i) متناظر با متن انتخابی (A_i,B_i) تنها با شرط $(A_i,B_i)\neq(A,B)$ دسترسی دارد، مؤثرترین راه برای به‌دست‌آوردن اطلاعات در مورد B (به‌جز طول آن) بازیابی کلید است.

۶-۲-۳- یک سامانه ایده‌آل

تعریف یک سامانه مرجع که این ویژگی‌ها را داشته باشد، با استفاده از یک زوج پیش‌گوی تصادفی (RO_1,RO_2) که رمزگذاری و محاسبه برچسب را به‌صورت زیر انجام می‌دهند، امکان‌پذیر است:

رمزگذاری: رمزگذاری با جمع بیتی B با دنباله کلید انجام می‌شود. دنباله کلید یادشده خروجی پیش‌گوی تصادفی RO_1 به رشته s_k است که از (K,A) با یک تابع کدگذاری یک به یک محاسبه شده است. اگر (K,A) در نقش تک‌شمار باشد، دنباله کلید برای داده‌های ورودی مختلف نتیجه فراخوانی RO_1 با ورودی‌های متفاوت بوده و بدین ترتیب، یک دنباله کلید اطلاعاتی در مورد سایر دنباله کلیدها افشا نخواهد کرد.

محاسبه برچسب: برچسب، خروجی پیشگوی تصادفی RO_2 به رشته h_t است که از (K,A,B) با یک تابع کدگذاری یک به یک محاسبه شده است. برچسب محاسبه‌شده برای پیام‌های مختلف نتیجه فراخوانی RO_2 با رشته ورودی متفاوت خواهد بود.

از آنجا که دنباله کلید و برچسب با فراخوانی پیشگوهای تصادفی متفاوتی تولید شده‌اند، اطلاعاتی از دیگری افشا نمی‌کنند؛ علاوه‌براین، از آنجا که کلید تنها به‌عنوان ورودی به پیش‌گوی تصادفی اعمال شده است، نیاز به عملی‌نبودن بازیابی کلید نیز برآورده می‌شود. دو پیش‌گوی تصادفی RO_1 و RO_2 با استفاده از روش تفکیک دامنه روی یک پیش‌گوی تصادفی RO قابل دست‌یابی است.

ساده‌ترین راه برای ساخت یک سامانه واقعی که به‌عنوان یک مدل مرجع عمل می‌کند، جایگزین پیش‌گوی تصادفی با تابع اسفنجی است؛ اما چنین راه‌کاری به اجرای دو تابع اسفنجی نیاز دارد، یکی برای تولید دنباله کلید و دیگری برای تولید برچسب. سبک عملکرد SPONGEWRAP راه‌کاری است که تنها به یک مرتبه فراخوانی به‌ازای هر قالب ورودی نیاز دارد.

۶-۲-۴- سبک عملکرد رمزنگاری احراز اصالت شده SPONGEWRAP

در این بخش به معرفی سبک عملکرد رمزنگاری احراز اصالت شده SPONGEWRAP می‌پردازیم. در اینجا نیز مشابه ساختار دوتایی، نمونه‌ای برای یک ساختار SPONGEWRAP را یک شیء SPONGEWRAP[5] نامیده و آن را با W نمایش می‌دهیم.

در مرحله مقداردهی اولیه به یک شیء SPONGEWRAP، کلید K در آن بارگذاری می‌شود؛ پس از آن می‌توان درخواست‌هایی برای پوشش یا واپیچی داده ارسال

[1] Wrapping
[2] Unwrapping
[3] Deterministic
[4] Nonce
[5] SPONGEWRAP object

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

کرد. قالب‌های دنباله کلید و برچسب به کلید K و داده‌های ارسال شده در تمامی درخواست‌های قبلی وابسته هستند. مراحل عنوان شده برای پوشش یا واپیچی داده تنها با یک درخواست در سبک عملکرد SPONGEWRAP قابل اجراست.

شیء SPONGEWRAP، W، از یک شیء دوتایی D استفاده می‌کند. در مقداردهی اولیه به یک شیء SPONGEWRAP، شیء یادشده D را مقداردهی اولیه کرده و قالب‌های کلید (دنباله‌زنی‌شده) را با فراخوانی صامت به D ارسال می‌کند.

درخواست پوشش با ورودی‌های سرآیند A، بدنه B و طول برچسب خروجی l با فراخوانی شیء SPONGEWRAP به‌صورت $W.wrap(A,B,l)$، ارسال می‌شود. متن رمزشده C قالب به قالب به‌صورت $C_i = B_i \oplus Z_i$ که در آن Z_i پاسخ D به فراخوانی قبلی $D.duplexing()$ است، تولید می‌شود. l بیت برچسب T، پاسخ D به آخرین قالب داده است (درصورتی‌که l بزرگ باشد، فراخوانی صامت نیز خواهیم داشت). در انتها متن رمزشده C و برچسب T به‌عنوان خروجی برگردانده می‌شود.

درخواست واپیچی با ورودی‌های سرآیند A، متن رمزشده C و برچسب T با فراخوانی شیء SPONGEWRAP به‌صورت $W.unwrap(A,C,T)$ دریافت می‌شود، سرآیند A به D ارسال می‌شود. بدنه B قالب به قالب به‌صورت $B_i = C_i \oplus Z_i$ رمزگشایی می‌شود که در آن Z_i پاسخ D به فراخوانی قبلی $D.duplexing()$ است. پاسخ D به آخرین قالب بدنه داده با برچسب T که به‌عنوان ورودی دریافت شده است مقایسه می‌شود. درصورتی‌که برچسب معتبر باشد، بدنه داده B برگردانده و در غیر این‌صورت خطا برگردانده می‌شود. توجه کنید که در زمان پیاده‌سازی می‌توان محدودیت‌هایی همچون فقط پیاده‌سازی پوشش یا واپیچی داده را نیز اعمال کرد؛ همچنین، شیء SPONGEWRAP می‌تواند محدودیتی را برای کمینه طول برچسب دریافت‌شده قبل از واپیچی در نظر بگیرد.

هر کلید، سرآیند، داده و یا متن رمزشده قبل از ارجاع به D با قاب بیتی[1] توسیع داده می‌شوند. اگر A، B و یا C رشته‌های تهی باشند، به‌صورت یک قالب شامل رشته تهی در نظر گرفته می‌شوند. نرخ ρ برای سبک عملکرد SPONGEWRAP تعیین کننده اندازه قالب‌ها و در نتیجه بیشترین تعداد بیت پردازش‌شده در هر بار فراخوانی f است.

[1] Farme bit

باند بالای نرخ برابر با $\rho_{max}(pad,r)-1$ است. تعریف SPONGEWRAP در الگوریتم (۵) (شکل ۹) آورده شده است.

Algorithm 5 The authenticated encryption mode SpongeWrap$[f,\text{pad},r,\rho]$.
Require: $\rho \le \rho_{max}(\text{pad},r)-1$
Require: $D = \text{duplex}[f,\text{pad},r]$
This algorithm treats A, B or C instances equal to the empty string as a single (empty) block

Interface: W.initialize(K) with $K \in \mathbb{Z}_2^+$
D.initialize()
for $i = 0$ to $|K|_\rho - 2$ **do**
  D.duplexing($K_i||1,0$)
end for
D.duplexing($K_{|K|_\rho-1}||0,0$)

Interface: $(C,T) = W$.wrap(A,B,ℓ) with $A, B \mathbb{Z}_2^*$, integer $\ell > 0$, $C \in \mathbb{Z}_2^{|B|}$ and $T \in \mathbb{Z}_2^\ell$
for $i = 0$ to $|A|_\rho - 2$ **do**
  D.duplexing($A_i||0,0$)
end for
$Z = D$.duplexing($A_{|A|_\rho-1}||1,|B_0|$)
$C = B_0 \oplus Z$
for $i = 0$ to $|B|_\rho - 2$ **do**
  $Z = D$.duplexing($B_i||1,|B_{i+1}|$)
  $C = C||(B_{i+1} \oplus Z)$
end for
$Z = D$.duplexing($B_{|B|_\rho-1}||0,\rho$)
while $|Z| < \ell$ **do**
  $Z = Z||D$.duplexing$(0,\rho)$
end while
$T = \lfloor Z \rfloor_\ell$
return (C,T)

Interface: $B = W$.unwrap(A,C,T) with $A,C \in \mathbb{Z}_2^*$, $T \in \mathbb{Z}_2^+$, $B \in \mathbb{Z}_2^{|C|} \cup \{\text{error}\}$
for $i = 0$ to $|A|_\rho - 2$ **do**
  D.duplexing($A_i||0,0$)
end for
$Z = D$.duplexing($A_{|A|_\rho-1}||1,|C_0|$)
$B_0 = C_0 \oplus Z$
for $i = 0$ to $|C|_\rho - 2$ **do**
  $Z = D$.duplexing($B_i||1,|C_{i+1}|$)
  $B_{i+1} = C_{i+1} \oplus Z$
end for
$Z = D$.duplexing($B_{|C|_\rho-1}||0,\rho$)
while $|Z| < \ell$ **do**
  $Z = Z||D$.duplexing$(0,\rho)$
end while
if $T = \lfloor Z \rfloor_\ell$ **then**
  return $B_0||B_1||\dots B_w$
else
  return Error
end if

(شکل-۹): الگوریتم SPONGEWRAP [۱]

قاب بیتی در الگوریتم (۵) برای دو هدف استفاده شده است:

تفکیک دامنه: ورودی‌های ساختار دوتایی برای تولید قالب‌های دنباله کلید و قالب‌های برچسب از دامنه‌های متفاوت هستند. هر پاسخ ساختار دوتایی که برای رمزگذاری قالب‌های پیام استفاده می‌شود، پاسخ به رشته ورودی است که به یک ختم می‌شود؛ درحالی‌که هر پاسخ ساختار دوتایی که برای تولید برچسب استفاده می‌شود، پاسخ به رشته ورودی است که به صفر ختم می‌شود.

کدگشایی: کلید، سرآیند و قالب‌های بدنه از دنباله ورودی به ساختار دوتایی قابل بازیابی است. این بدان معنی است که دو رشته متفاوت $K, A^{(0)}, B^{(0)}, A^{(1)}, B^{(1)}, \dots$ و $K', A'^{(0)}, B'^{(0)}, A'^{(1)}, B'^{(1)}, \dots$ دو رشته ورودی یکسان به ساختار دوتایی را نتیجه نمی‌دهند.

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]
دوفصل‌نامه علمی ترویجی اطلاعات تبادل تولید و امنیت فضای

مزایا و محدودیت‌ها:

مزایای سبک عملکرد رمزنگاری احراز اصالت‌شده SPONGEWRAP عبارتند از:

- این سبک عملکرد تنها به طراحی یک جایگشت با طول ثابت نیاز دارد، برخلاف برخی از سبک‌های عملکرد رمزنگاری احراز اصالت‌شده که به طراحی یک رمز قالبی نیاز دارند.
- این سبک عملکرد علاوه بر رمزنگاری احراز اصالت‌شده، برای کاربردهایی که در آنها فقط به احراز اصالت نیاز است، نیز قابل استفاده است.
- قابلیت تولید برچسب‌های میانی وجود دارد.
- یک طرفه است.
- برای هر قالب ورودی تنها یکبار فراخوانی تابع f مورد نیاز است.
- انتخاب نرخ بیتی متنوع، مادامیکه ظرفیت از مقدار معینی کمتر نباشد، از ویژگی‌های این سبک عملکرد است.

در مقایسه با سبک‌های عملکرد رمزنگاری احراز اصالت‌شده مبتنی بر رمزهای قالبی، SPONGEWRAP دارای محدودیت‌هایی نیز هست.

محدودیت‌های سبک عملکرد رمزنگاری احراز اصالت‌شده SPONGEWRAP عبارتند از:

- این سبک عملکرد به‌صورت سریال بوده و قابلیت پیاده‌سازی موازی در سطح الگوریتم را ندارد.
- اگر سامانه محدودیتی به‌عنوان الزام استفاده از تک‌شمار در A را نداشته باشد، مهاجم می‌تواند دو پرسمان (A,B) و (A,B') را انجام دهد. در این‌صورت، نخستین قالب متفاوت در B، به‌عنوان مثال B_i و B'_i، با کلید یکسانی رمز شده و اطلاعاتی در مورد جمع بیتی آنها را برای مهاجم افشا می‌کند. این در حالی است که برخی از سبک‌های عملکرد رمزنگاری احراز اصالت‌شده برای رمزهای قالبی به‌گونه‌ای طراحی شده‌اند که حتی درصورتی‌که سامانه شرط استفاده از تک‌شمار را منظور نکند، تنها اطلاعاتی که برای مهاجم افشا می‌شود، تساوی یا عدم تساوی دو ورودی به الگوریتم است [۲۰].

کاربرد:

پوشش کلید عبارتست از حفظ محرمانگی و تمامیت کلید در انتقال یا ذخیره آن. کلید اصلی با کلید رمزگذاری کلید[1](KEK) پوشش داده می‌شود. می‌توان از سبک عملکرد SPONGEWRAP با KEK به‌عنوان K و کلید اصلی به‌عنوان بدنه داده استفاده کرد. در سامانه‌های مدیریت کلید، هر کلید یک شناسه منحصر به‌فرد دارد. بدین ترتیب، کافیست شناسه هر کلید در سرآیند A قرار داده شود. در این صورت، دو کلید اصلی هیچگاه با یک دنباله کلید رمز نمی‌شود. در پوشش کلید، می‌توان از کلید عمومی متناظر و یا چکیده محاسبه‌شده از کلید مورد نظر به‌عنوان شناسه استفاده کرد.

[1] Key-encrypting key

۶-۳- تولید دنباله اعداد شبه‌تصادفی با امکان بارگذاری مجدد بذر

اعداد شبه‌تصادفی کاربردهای فراوانی در رمزنگاری دارند. تولید کلید یکی از کاربردهای متداول این‌گونه دنباله از اعداد است. اگرچه خاصیت تصادفی‌بودن از یک منبع فیزیکی قابل استخراج است، اما اغلب به تعداد بیت‌هایی بیشتر از آنتروپی یک منبع فیزیکی نیاز است. یک مولد دنباله اعداد شبه‌تصادفی راه حلی برای این نیاز است. چنین مولدی با یک بذر[2] که به‌صورت مخفی و یا به‌طورکامل تصادفی تولید شده بارگذاری شده و آن را به دنباله‌ای از بیت‌ها توسیع می‌دهد.

یکی از ویژگی‌های مهم برای دنباله اعداد شبه‌تصادفی در کاربردهای رمزنگاری آن است که حتی در صورت افشای بخشی از دنباله، پیش‌گویی سایر قسمت‌های آن ممکن نباشد. از این جنبه، مولد اعداد شبه‌تصادفی مشابه یک رمز دنباله‌ای است؛ علاوه‌براین، در برخی کاربردها ویژگی امنیت پیشرو[3] بایستی فراهم شود. این ویژگی بدان معناست که آشکارشدن حالت داخلی کنونی امکان آشکارکردن بیت‌های قبلی تولیدشده را فراهم نکند.

حالت داخلی مولد اعداد شبه تصادفی بایستی آنتروپی لازم را داشته باشد. این بدان معناست که پیش‌بینی بیت‌های خروجی به‌سادگی با حدس حالت داخلی ممکن نباشد؛ بنابراین، دنباله‌ای که به‌عنوان بذر مورد استفاده قرار می‌گیرد، بایستی آنتروپی کافی را فراهم کند. دنباله تولیدشده به‌عنوان بذر با استفاده از منابع فیزیکی به‌عنوان مولد خاصیت تصادفی، بدلیل نامتوازن‌بودن و یا همبستگی بیت‌ها، عموما آنتروپی بالایی ندارند. برای افزایش آنتروپی، می‌توان از بذرهای مختلف که از منابع تصادفی مختلف به‌دست آمده استفاده کرد؛ اما این آنتروپی بایستی به حالت داخلی محدود مولد اعداد شبه‌تصادفی منتقل شود؛ بنابراین به روشی برای ترکیب آنتروپی تولید شده از منابع مختلف در حالت داخلی مولد نیاز داریم. بارگذاری بذرهای مختلف در مولد اعداد شبه‌تصادفی دنباله‌های متفاوتی را نتیجه می‌دهد. این بدان معناست که

[2] Seed
[3] Forward secrecy

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

بذرهای مختلف حالت‌های داخلی مختلف را نتیجه می‌دهند؛ از این جنبه، یک مولد اعداد شبه تصادفی مانند یک تابع چکیده‌ساز مقاوم در برابر برخورد است.

تحقق امکان بارگذاری مجدد بذر برای یک مولد اعداد شبه‌تصادفی ساده است. برای این منظور، کافی است امکان ورود بذرهای اضافه بعد از تولید بیت‌های شبه‌تصادفی فراهم شود. بارگذاری مجدد بذر، به‌جای حذف‌کردن حالت داخلی کنونی، آن را با بذر جدید ترکیب می‌کند. از دیدگاه کاربر، یک مولد اعداد شبه‌تصادفی با امکان بارگذاری مجدد بذر به‌صورت یک جعبه سیاه با یک واسط برای درخواست دنباله بیت‌های شبه‌تصادفی و یک واسط برای ورودی بذرهای جدید است. در ادامه، منظور از مولد اعداد شبه‌تصادفی مولد اعداد شبه‌تصادفی با امکان بارگذاری مجدد بذر است.

۶-۳-۱- مدل‌کردن یک مولد شبه‌تصادفی ایده‌آل

مولد شبه‌تصادفی را به‌صورت یک واحد حافظه‌دار با پشتیبانی از دو نوع درخواست تعریف می‌کنیم.

- درخواست تغذیه[1]، $feed(\sigma)$، که رشته ناتهی $\sigma \in Z_2^+$ حاوی بذر را به حالت داخلی مولد شبه‌تصادفی تزریق می‌کند.
- درخواست واکشی[2]، $fetch(l)$، که برگرداندن l بیت را از مولد شبه‌تصادفی درخواست می‌کند.

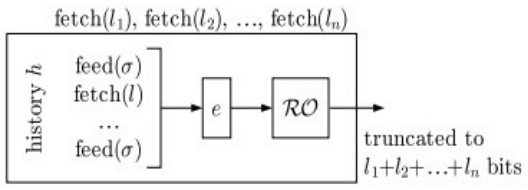
دادهٔ تشکیل‌دهنده بذر حاصل الحاق σ ها در تمامی درخواست‌های تغذیه است.

نیازمندی‌های یک مولد شبه‌تصادفی بدین‌صورت قابل بیان است: نخست، خروجی آن بایستی به تمامی دادهٔ تشکیل‌دهندهٔ بذر وابسته باشد. دوم، برای مهاجمی که از بذر آگاهی نداشته و بخشی از خروجی را مشاهده کرده است، یافتن اطلاعاتی در مورد سایر بخش‌های خروجی غیرعملی باشد.

بیان دقیق‌تر نیازمندی‌های امنیتی یک مولد شبه‌تصادفی اغلب با تعریف یک سامانه مرجع ایده‌آل صورت می‌گیرد. سامانه مرجع مناسب برای توابع اسفنجی، توابع چکیده‌ساز و رمزهای دنباله‌ای پیش‌گوی تصادفی است؛ اما از آنجا که پیشگوی تصادفی واسط متفاوتی با مولد شبه‌تصادفی با امکان بارگذاری مجدد بذر دارد، نمی‌توان از آن به‌عنوان سامانه مرجع مناسبی برای مولد شبه‌تصادفی استفاده کرد. با این حال، یک مولد شبه‌تصادفی ایده‌آل را به‌صورت سبک عملکرد خاصی از پیش‌گوی تصادفی تعریف می‌کنیم.

[1] Feed
[2] Fetch

سبک عملکرد یادشده تمامی درخواست‌های تغذیه و واکشی را در حافظه h نگه می‌دارد. با دریافت درخواست $feed(\sigma)$ ، حافظه h به‌روزرسانی می‌شود. با دریافت درخواست $fetch(l)$، یک پرسمان از پیش‌گوی تصادفی با رشته‌ای که حافظه h را کدگذاری کرده صورت می‌گیرد. پاسخ پیش‌گوی تصادفی از بیت x تا بیت x+l-1 به‌عنوان خروجی به پرسش‌گر برگردانده می‌شود، که در آن x تعداد بیت‌های درخواستی در درخواست‌های واکشی قبلی از زمان آخرین درخواست تغذیه است. بنابراین، پیوست پاسخ‌ها به دنباله‌ای از درخواست‌ها معادل با پاسخ پیش‌گوی تصادفی تنها به یک پرسمان است. این وضعیت در شکل (۱۰) نشان داده شده است. سبک عملکرد یادشده، سبک عملکرد حافظه‌دار با تابع کدگذاری $e(h)$ نامیده می‌شود؛ بنابراین، تعریف سبک عملکرد مورد نظر به تعریف این تابع کدگذاری کاهش می‌یابد.



(شکل-۱۰): پاسخ یک مولد شبه‌تصادفی به درخواست‌های واکشی

از آنجا که خروجی مولد شبه‌تصادفی بایستی به تمامی داده بذر وابسته باشد، لازم است تا تابع کدگذاری یک‌به‌یک باشد. به بیان دیگر، برای دو دنباله از درخواست‌ها با بذرهای مختلف، دو خروجی متفاوت از تابع کدگذاری بایستی داشته باشیم. این ویژگی تمامیت بذر[3] نامیده می‌شود.

تعریف ۱۳: یک مولد شبه‌تصادفی ایده‌آل یک پیشگوی تصادفی در سبک عملکرد حافظه‌دار با تابع کدگذاری $e(h)$ با ویژگی تمامیت بذر است.

گفتنی است که مولد شبه‌تصادفی مورد نظر فاقد ویژگی امنیت پیشرو است (افشای حالت داخلی کنونی مهاجم را قادر به بازیابی بیت‌های خروجی قبلی نمی‌کند).

۶-۳-۲- SpongePRG

یک روش ساده برای ساخت یک سامانه واقعی که مانند سامانه مرجع فوق عمل می‌کند، جایگزین‌کردن پیش‌گوی تصادفی با تابع اسفنجی است. در نگاه نخست، نیاز به ذخیره تمامی پرسمان‌های قبلی مستلزم حافظه زیاد بوده و این راه‌کار عملی به نظر نمی‌رسد؛ اما با استفاده از ساختار دوتایی این مانع رفع می‌شود.

[3] Seed-completness

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

یک شیء دوتایی به‌عنوان یک مولد شبه‌تصادفی با امکان بارگذاری مجدد بذر قابل استفاده است. درخواست تغذیه با ورودی σ با فراخوانی $D.duplexing()$ صورت گرفته و خروجی به‌عنوان بیت‌های شبه‌تصادفی قابل استفاده است. بیت‌های شبه‌تصادفی بدون داشتن بذر با فراخوانی خالی $D.duplexing()$ قابل تولید است. تنها محدودیت این ساختار آن است که رشته حاوی بذر بایستی به طول ρ_{max} بریده شده و نیز در هر بار فراخوانی تنها r بیت خروجی قابل تولید است.

سبک عملکرد SpongePRG به‌صورت زیر کار می‌کند. این سبک عملکرد از یک شیء D استفاده کرده و دارای دو بافر است: بافر ورودی B_{in} و بافر خروجی B_{out}. بذر در درخواست‌های تغذیه به B_{in} فرستاده شده و با فراخوانی $D.duplexing()$ به شیء D ارسال می‌شود. درصورتی‌که تعداد بیت‌های بذر بیشتر از ρ_{max} باشد، بیت‌های اضافی در B_{in} می‌ماند. با دریافت درخواست واکشی، در صورت خالی‌نبودن بافر ورودی، بیت‌های باقی مانده از بذر به D ارسال شده و بیت‌های درخواست‌شدهٔ خروجی داده می‌شود (در صورت نیاز تعداد دفعات فراخوانی $D.duplexing()$ قابل افزایش است). بیت‌های اضافی خروجی در بافر خروجی نگهداری شده و در درخواست واکشی بعدی ابتدا این بیت‌ها برگردانده می‌شود. توجه کنید که در هر زمان یکی از بافرهای ورودی و یا خروجی خالی است.

Algorithm 6 Pseudo random bit sequence generator mode SpongePRG$[f, \text{pad}, r, \rho]$

Require: $\rho \leq \rho_{\max}(\text{pad}, r)$
Require: $D = \text{DUPLEX}[f, \text{pad}, r]$

Interface: P.initialize()
D.initialize()
B_{in} = empty string
B_{out} = empty string

Interface: P.feed(σ) with $\sigma \in \mathbb{Z}_2^+$
$M = B_{\text{in}} || \sigma$
for $i = 0$ to $|M|_\rho - 2$ **do**
  D.duplexing(M_i, 0)
end for
$B_{\text{in}} = M_{|M|_\rho - 1}$
B_{out} = empty string

Interface: $Z = P$.fetch(ℓ) with integer $\ell \geq 0$ and $Z \in \mathbb{Z}_2^\ell$
while $|B_{\text{out}}| < \ell$ **do**
  $B_{\text{out}} = B_{\text{out}} || D$.duplexing($B_{\text{in}}, \rho$)
  B_{in} = empty string
end while
$Z = \lfloor B_{\text{out}} \rfloor_\ell$
B_{out} = last ($|B_{\text{out}}| - \ell$) bits of B_{out}
return Z

Interface: P.forget()
$Z = D$.duplexing(B_{in}, ρ)
B_{in} = empty string
for $i = 1$ to $\lfloor c/\rho \rfloor$ **do**
  $Z = D$.duplexing(Z, ρ)
end for
B_{out} = empty string

(شکل-۱۱): الگوریتم سبک عملکرد **SpongePRG** [Error! Bookmark not defined.

اگر f یک جایگشت باشد، SpongePRG برگشت‌پذیر خواهد بود و افشای حالت داخلی، امکان برگشتن به حالت داخلی در آخرین اعمال بذر را فراهم می‌کند. البته راه‌کارهایی برای این مشکل وجود دارد که نمونه‌ای از آن در [۱] معرفی شده است. سبک عملکرد SpongePRG در الگوریتم (۶) (شکل ۱۱) تعریف شده است.

مزایا و محدودیت‌ها:

ایده اصلی مبنی بر ترکیب منابع مختلف بذر و تولید بیت‌های شبه‌تصادفی است. تنها نیازمندی برای داده بذر آن است که به‌صورت دنباله‌ای از بیت‌ها باشد، که بدون هیچ‌گونه پیش پردازش اضافی در دسترس است. بنابراین، اعمال بذر و تولید بیت‌های شبه‌تصادفی در یک سبک عملکرد پیوسته قابل پیاده‌سازی بوده و برای اعمال داده بذر اضافی نیازی به تکرار اضافه‌تر تابع به‌روزرسانی نیست.

اگر f یک جایگشت باشد، طول حالت داخلی را می‌توان کوتاه نگه داشت. این امکان از دو ویژگی زیر فراهم می‌شود. ویژگی نخست، حفظ آنتروپی حالت داخلی با استفاده از جایگشت و ویژگی دوم، وجود باند بالای قوی برای احتمال موفقیت حملات عمومی روی اسفنج کلیددار است.

عدم تأمین ویژگی امنیت پیشرو در ساختار معرفی‌شده ضعف آن به‌شمار می‌رود. این ضعف به دو دلیل در بسیاری از کاربردها مشکل‌ساز نیست؛ نخست آن‌که اعمال بذر به‌صورت منظم و با آنتروپی کافی مانع پیشروی مهاجم در جهت عقب می‌شود. دوم آن‌که تجهیزی که مولد شبه‌تصادفی مورد نظر روی آن پیاده می‌شود بایستی محرمانگی داده را حفظ کرده و از این رو بازخوانی حالت داخلی مشکل خواهد بود. نمونه‌ای از استفاده از ساختار اسفنج برای تولید اعداد شبه‌تصادفی در [۵] معرفی شده است.

۷- نتیجه‌گیری

در این نوشتار به معرفی ساختار اسفنجی که در سال‌های اخیر به یکی از ساختارهای پرکاربرد در طراحی اولیه‌های رمزنگاری تبدیل شده پرداختیم. برای این منظور، ساختار اسفنجی پایه و ساختار دوتایی را تعریف و حملات اولیه روی آنها را بررسی کردیم. در ادامه به تشریح نحوه استفاده از اسفنج در طراحی توابع چکیده‌ساز، الگوریتم‌های رمزنگاری احراز اصالت شده و

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.1.2.8]

مولد اعداد شبه‌تصادفی پرداخته و مزایا و محدودیت‌های هر کدام را مورد ارزیابی قرار دادیم.

۸- مراجع

[1] G.Bertoni, J.Daemen, M.Peeters, and G. V.Assche, (2011), "Cryptographic Sponge Functions", Retrieved from http://sponge.noekeon.org/.

[2] G.Bertoni, J. Daemen, M. Peeters, and G. V.Assche, (2013). Keccak. EUROCRYPT 2013, LNCS 7881, pp. 313-314.

[3] W.Li, G. Liao, Y.Wen, and Z.Gong, "SpongeMPH: A New Multivariate Polynomial Hash Function based on the Sponge Construction", *Second International Conference on Data Science in Cyberspace (DSC).Shenzhen: IEEE*, (2017).

[4] N.*Abdoun, S. El Assad, K. Hammoud, R.Assaf, M.Khalil, and O. Diforges, "New Keyed Chaotic Neural Network Hash Function Based on Sponge Construction", International Conference for Internet Technology and Secured Transactions (ICITST). Cambridge: IEEE*, (2018).

[5] P.Kumar Singh, A. V. Monsy, R. Garg, S.Dey, and S.Nandi, "JSpongeGen: A Pseudo Random Generator for Low Resource Devices", I*nternational Conference on Distributed Computing and Internet Technology, ICDCIT 2019*, pp. 410-421, 2019.

[6] R.AlTawy, R.Rohit, M. He, K.Mandal, G.Yang, and G.Guang, "sLiSCP: Simeck-Based Permutations for Lightweight Sponge Cryptographic Primitives", *International Conference on Selected Areas in Cryptography. LNCS 10719*, pp. 129-150 , 2018.

[7] G.Bertoni, J.Daemen, G. V.Assche, and M.Peeters, Radiogatun, a Belt-and-Mill Hash Function, Second Cryptographic Hash Workshop, Available: http://radiogatun.noekeon.org/. [Accessed:2006].

[8]P.Jovanovic, A. Luykx, and B.Mennink, "Beyond 2^c/2 Security in Sponge-Based Authenticated Encryption Modes", *ASIACRYPT 2014*, *LNCS 8873*, pp. 85-104. 2014.

[9] B.Mennink, R.Reyhanitabar, and D.Vizar," Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption", *ASIACRYPT 2015, LNCS 9453*, pp. 465-489. 2015.

[10] Y.Naito and K.Yasuda, "New Bounds for Keyed Sponges with Extendable Output: Independence Between Capacity and Message Length", *FSE 2016, LNCS 9783*, pp. 3-22, Berlin, 2016.

[11] L.Song and J.Guo, "Cube-Attack-Like Cryptanlysis of Round-Reduced Keccak Using MILP", *IACR Transactions on Symmetric Cryptology*, pp. 182-214, 2018.

[12] R.Kumar, M. S. Rajasree, and H. Alkhzaimi, "Cryptanalysis of 1-Round KECCAK. AFRICACRYPT 2018", *LNCS 10831*, pp. 124-137, 2018.

[13] G.Bertoni, J. Daemen, M. Peeters, and G. V Assche, "Sufficient Conditions for Sound Tree and Sequential Hashing Modes", *International Journal of Information Security*, pp.335-353, 2014.

[14] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. Retrieved from http://competitions.cr.yp.to/caesar.html/.

[15] M.Bellare and C.Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm", *ASIACRYPT 2000*, *LNCS 1976*, pp. 531-545, Berlin,2000.

[16] V. D. Gligor and P. Donescu, "Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes", FSE 2001, *LNCS 2355*, pp. 92-108. Berlin, 2001.

[17] C. S. Jutla, "Encryption Modes with Almost Free Message Integrity", *EUROCYPT 2001, LNCS 2045,* pp. 529-544. Berlin,2001.

[18] P.Rogaway, M. Bellare, J.Black, and T.Krovetz, "OCB: A Block Cipher Mode of Operation for Efficient Authenticated Encryption", *ACM* Conference on Computer and Communication Security, pp. 196-205, 2001.

[19] NIST, Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. NIST special publication 800-38C. 2007

[20] P.Rogaway and T.Shrimpton, "A Provable-Security Treatment of the Key-Wrap Problem", *EUROCRYPT 2006*. LNCS 4004, pp. 373-390. Berlin: Springer, 2006.

[Downloaded from monadi.isc.org.ir on 2026-11-20]

[DOR: 20.1001.1.24763047.1398.8.1.2.8]

[DOR: 20.1001.1.24763047.1398.8.1.2.8] [Downloaded from monadi.isc.org.ir on 2026-11-20]