مروری بر پیاده‌سازی آستانه‌ای به‌عنوان روشی برای مقابله با حملات تحلیل توان

سارا زارعی[1] و هادی سلیمانی*[2]

[1] پژوهشکده فضای مجازی، دانشگاه شهید بهشتی، تهران، ایران
s.zareei@mail.sbu.ac.ir

[2] استادیار پژوهشکده فضای مجازی، دانشگاه شهید بهشتی، تهران، ایران
h_soleimany@sbu.ac.ir

چکیده

از جمله روش‌های کشف اطلاعات حساس و پارامترهای مخفی الگوریتم‌های رمزنگاری، استفاده از داده‌هایی است که به‌صورت فیزیکی از دستگاه‌ها نشت می‌یابند. حملات تحلیل توان، از نوع حملات کانال جانبی بوده و بسیار کاربردی هستند؛ بنابراین اتخاذ روش‌هایی به‌منظور مقاوم‌سازی ابزارهای رمزنگاری در برابر این‌گونه حملات از اهمیت بالایی برخوردار است. روش‌های مقاوم‌سازی می‌توانند از رویکردهای مختلفی استفاده کرده و در سطوح متفاوتی اعمال شوند. یکی از روش‌های مؤثر و متداول، روش نقاب‌گذاری است که در سطح الگوریتم، امنیت اثبات‌پذیر ایجاد می‌کند. با این حال می‌توان نشان داد که برخی از الگوریتم‌های نقاب‌گذاری‌شده نیز، در زمان پیاده‌سازی سخت‌افزاری دچار گونه‌هایی از نشت اطلاعات هستند که گاهی آن‌ها را در برابر حملات تحلیل توان آسیب‌پذیر می‌سازد. روش پیاده‌سازی آستانه‌ای روشی است که به‌منظور مواجهه با این چالش معرفی شده و طی سالیان اخیر توجه بسیاری از پژوهش‌گران را به خود جلب کرده است. در این مقاله ابتدا مدل‌های مختلف حملات وارده به رمزهای قالبی را توضیح داده و سپس با تمرکز بر مدل جعبه خاکستری و حملات تحلیل توان، روش نقاب‌گذاری را شرح می‌دهیم. در ادامه با بیان چالش این روش در پیاده‌سازی‌های سخت‌افزاری، روش پیاده‌سازی آستانه‌ای را معرفی و بررسی خواهیم کرد.

واژگان کلیدی: رمزهای قالبی، امنیت سخت‌افزاری، پیاده‌سازی آستانه‌ای

۱- مقدمه

حملات بازیابی کلید، از انواع مهم حملات ارائه‌شده به الگوریتم‌های رمزنگاری محسوب می‌شوند. در این سناریو مهاجم تلاش می‌کند با توجه به میزان اطلاعاتی از الگوریتم که می‌تواند به آن‌ها دسترسی داشته باشد، حمله خود را اجرایی کند. بر همین اساس، سطوح دسترسی مختلفی که یک مهاجم می‌تواند به دستگاه مورد هدف داشته باشد، حملات متفاوتی را محتمل می‌سازند که آن‌ها را در سه دستهٔ کلی طبقه‌بندی و تعریف می‌کنیم.

۱-۱- مدل جعبهٔ سیاه

در این مدل مواجهه مهاجم با الگوریتم رمزنگاری تنها به مثابه یک تابع معلوم ریاضی است که به ورودی‌ها (متون اصلی) و خروجی‌ها (متون رمزشده) یی از آن دسترسی دارد و تنها با دانستن این اطلاعات و بدون دسترسی داشتن به مقادیر میانی و پردازش‌های داخلی، باید کلید مخفی را با پیچیدگی کمتر از جستجوی‌کامل[1] استخراج کند. چنین حمله‌ای فارغ از نحوهٔ پیاده‌سازی است و تنها مبتنی بر الگوریتم و ضعف‌های احتمالی آن است. تحلیل‌هایی از جمله تحلیل‌های خطی[2] و تفاضلی[3] مهم‌ترین و مشهورترین حملاتی هستند که در این مدل جای می‌گیرند [1].

گفتنی است الگوریتم‌های استانداردی که امروزه در پیاده‌سازی‌ها به‌کار می‌روند در برابر چنین حملاتی امن‌سازی شده‌اند.

۱-۲- مدل جعبهٔ سفید

در این مدل برخلاف مدل قبلی، مهاجم به تمام جزییات و مراحل پیاده‌سازی از جمله مقادیر میانی تحت پردازش،

[1] Exhaustive search
[2] Linear cryptanalysis
[3] Differential cryptanalysis

*نویسندهٔ مسؤول

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.2.2.0]

منابع حافظه، مراحل اجرای الگوریتم و... اشراف و کنترل دارد. به‌عبارتی تمام اطلاعات به‌جز کلید مخفی الگوریتم، در دسترس مهاجم قرار خواهند داشت [2]. قابل پیش‌بینی است که درصورت لحاظ‌نکردن ملاحظات امنیتی مرتبط، چنین حملاتی می‌توانند در محیط‌هایی هم‌چون تلفن‌های هوشمند[1]، سرورها و ... بسیار خطرآفرین باشند که البته از حوزه بررسی ما در این مقاله خارج است.

۳-۱- مدل جعبهٔ خاکستری

در مدل جعبهٔ خاکستری تمرکز مهاجم بر پیاده‌سازی الگوریتم رمزنگاری و سوءاستفاده از اطلاعاتی است که به‌موجب پیاده‌سازی نشت می‌یابند؛ از قبیل زمان اجرای الگوریتم‌ها[2] [3]، تشعشعات الکترومغناطیسی[3] [4] و مصرف توان[4] [5]. به این ترتیب مهاجم علاوه‌بر داده‌های ورودی و خروجی و خود الگوریتم، اطلاعاتی در اختیار خواهد داشت که می‌توانند درمجموع منجر به یافتن کلید یا مقادیر حساس شوند؛ بنابراین فرض اولیه در این مدل حملات این است که مهاجم به ابزار مورد هدف دسترسی فیزیکی دارد و می‌تواند چنین اطلاعات نشتی را از آن به‌دست آورده و ثبت کند. رویکرد به‌دست‌آوردن این اطلاعات می‌تواند به‌صورت تهاجمی[5] [6]، نیمه‌تهاجمی[6] [7] و یا غیرتهاجمی[7] [3] [4] [5] باشد. به‌عنوان یک دسته‌بندی دیگر، حملات به یکی از دو دسته فعال یا غیرفعال تقسیم می‌شوند. حملات تحلیل توان در دستهٔ غیرتهاجمی و غیرفعال (حملات کانال جانبی) قرار می‌گیرند. این دسته از سایر انواع کاربردی‌تر، ارزان‌تر و موثرتر هستند.

۲- حملات تحلیل توان

تحلیل توان می‌تواند به دو صورت ساده[8] [5] و تفاضلی[9] [5] انجام شود. در حملات تحلیل توان ساده، مهاجم با دادن ورودی‌های یکسان به دستگاه، الگو[10](ها)یی از مصرف توان به‌دست آورده و با تحلیل آن(ها) به مقدار مطلوب می‌رسد؛ با این حال این روش در عین سادگی و اختصار نسبت به روش تفاضلی، ضعف‌هایی نیز دارد که در برخی از موارد اجرای آن را بی‌فایده می‌سازد؛ از جمله این‌که در صورت وجود نوفه (پایین‌بودن SNR[11]) مهاجم نخواهد توانست الگوهای دقیقی از مصرف توان به‌دست آورد؛ هم‌چنین در صورت ندانستن جزییات پیاده‌سازی ابزار مورد هدف و یا عدم دسترسی به ابزاری به‌طوردقیق مشابه به‌منظور به‌دست‌آوردن الگوهای مصرف توان (فاز پروفایلینگ[12] یا آفلاین)، مهاجم امکان مقایسه و تحلیل نخواهد داشت.

در مقابل، تحلیل تفاضلی به‌علت عدم وابستگی به جزییات پیاده‌سازی کاربردی‌تر است. این حمله با توجه به ممان[13] آماری که به‌عنوان ابزار مقایسهٔ الگوها استفاده می‌کند (تفاضل، واریانس، انحراف از معیار و...)، می‌تواند روابط میان الگوهای توان و درنتیجه مقادیر میانی حساس یا وابسته به کلید و یا حتی خود کلید را کشف کند؛ بنابراین هر الگوریتمی حتی اگر در مدل جعبهٔ سیاه استاندارد و امن باشد، درصورتی‌که با روشی غیرمقاوم در مقابل تحلیل توان پیاده‌سازی شود، در معرض آسیب خواهد بود.

۳- روش‌های مقاوم‌سازی

با توجه به توضیحات داده‌شده، طراحی و ارزیابی روش‌هایی برای پیاده‌سازی امن رمزهای قالبی، به‌گونه‌ای که در برابر حملات تحلیل توان ایمن باشند، از اهمیت بالایی برخوردار است. این روش‌ها در سطوح مختلفی ارائه و به‌کار گرفته شده‌اند که از سطوح بالای سیستمی[14] تا سطح دستورالعمل‌ها[15] و سلول‌های‌منطقیِ[16] مورد استفاده در پیاده‌سازی را شامل می‌شوند.

- **دسته نخست**

روش‌هایی که تلاش می‌کنند بر الگوریتم تمرکز کرده و آن را به‌گونه‌ای محدود سازند که تعداد عملیات‌هایی که با یک کلید یکسان انجام می‌شوند، برای انجام حمله تحلیل توان توسط مهاجم کافی نباشند [8] [9]. بدیهی است، چنین روش‌هایی با تحمیل هزینه‌های فراوان به‌منظور تأمین کلیدهای متعدد همراه خواهند بود. این هزینه‌ها بر عملکرد سیستم تأثیر گذاشته و موجب افت شدید آن می‌شوند. کاهش عملکرد سیستم در کاربردهای بسیاری به‌ویژه در تراشه‌های کم‌حجم و سبک قابل قبول و مطلوب نیست.

[1] Smart phone
[2] Execution time
[3] Electromagnetic (EM) Emission
[4] Power consumption
[5] Invasive
[6] Semi-Invasive
[7] Non-Invasive
[8] Simple Power Analysis
[9] Differential Power Analysis
[10] Trace
[11] Signal-to-Noise Ratio
[12] Profiling
[13] Moment
[14] System level
[15] Instruction level
[16] Cell level

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.2.2.0]

• دسته دوم

در این دسته رویکرد و هدف کلی کاهش بیشینه‌ای ارتباط میان مقادیر میانی با مقادیر حساس و مخفی است. هرچند که این روش‌ها در موارد متعددی مؤثر و کارآمد هستند، با این‌حال با افزایش زمان حمله، توان محاسباتی و تحلیلی، مهاجم قادر است، درنهایت بر آن‌ها غلبه کند [10] [11] [12].

• دسته سوم

رویکردهایی که به‌دنبال حذف تناظر میان مصرف توان الگوریتم و مقادیر حساس و مخفی هستند. به این ترتیب مهاجم هر تعداد الگوی مصرف توان را هم که از ابزار به‌دست آورد، با تحلیل آن‌ها نخواهد توانست به مقادیر میانی واقعی و درنتیجه پارامترهای حساس و مخفی برسد. به این روش، نقاب‌گذاری[1] گفته می‌شود که در آن با استفاده از طرح‌های تسهیم راز[2]، مقادیر میانی به‌گونه‌ای تصادفی‌سازی می‌شوند که الگوی توان به‌دست‌آمده ارتباط مستقیمی با داده‌های واقعی نداشته باشد.

۳-۱- روش نقاب‌گذاری

نخستین طرح‌های نقاب‌گذاری پیشنهادی با استفاده از قواعد جبر بولین[3]، هر مقدار میانی X را با دو مقدار تصادفی X_1 و X_2 نقاب‌گذاری می‌کنند؛ به‌صورتی‌که رابطه (۱) برقرار باشد:

$$\boldsymbol{X_1} \oplus \boldsymbol{X_2} = \mathrm{X} \qquad (۱)$$

X_1 به‌صورت تصادفی انتخاب و X_2 به‌نحوی تنظیم می‌شود که XOR آن‌ها برابر با مقدار داده اصلی یعنی X باشد. به هر یک از مقادیر X_1 و X_2 یک سهم گفته می‌شود.

مهاجم تنها درصورتی می‌تواند مقدار X را پیدا کند که مقادیر تصادفی X_1 و X_2 را بداند. این اتفاق درعمل غیرممکن است؛ چون که X_1 و X_2 تصادفی هستند؛ اما فرض کنیم که مهاجم از حمله کانال جانبی استفاده کرده و بخواهد با تحلیل الگوهای مصرف توان، به اطلاعات نشتی دسترسی پیدا کند که در پیدا کردن X مؤثر باشند.

اگر اطلاعات نشستی را با استفاده از وزن همینگ[4] مدل کنیم، مهاجم با میانگین‌گرفتن از داده‌های نشتی به اطلاعاتی در مورد X نخواهد رسید؛ اما با اعمال واریانس، همان‌طورکه در جدول (۱) می‌بینیم، نتایج به‌دست‌آمده به‌نحوی هستند که به‌ازای مقادیر مختلف X متفاوت‌اند؛ بنابراین مهاجم می‌تواند بدون نیاز به پیداکردن دقیق X_1 و X_2، مقدار داده X را به‌دست آورد.

(جدول-۱): الگوی نشت اطلاعات در نقاب‌گذاری درجه نخست داده یک بیتی

X	X_1	X_2	$L(X)$	$Mean(L(x))$	$Var(L(x))$
0	0	0	0	1	1
	1	1	2		
1	0	1	1	1	0
	۱	۰	۱		

۳-۲- نقاب‌گذاری مرتبهٔ بالا

به‌طورکلی، یک طرح نقاب‌گذاری مرتبه dام، یک روش ایمن‌سازی در برابر حمله تحلیل توان مرتبه d است که در آن هر مقدار میانی به $d+1$ سهم تقسیم و سهم‌ها به‌صورت تصادفی و یکنواخت[5] انتخاب می‌شود. مهاجمی با توان مرتبه dام، تنها قادر است که به d مقدار از این $d+1$ مقدار دسترسی پیدا کند و با داشتن این مقادیر، قادر به بازسازی مقدار میانی مورد نظر نخواهد بود.

۳-۳- چالش احتمالی روش نقاب‌گذاری در پیاده‌سازی سخت‌افزاری

در طرح‌های نقاب‌گذاری موجود [13] [14] [15] [16] [17]، فرض می‌شود که تغییرات متغیرهای میانی به‌صورت پایدار بوده و تغییرات کوتاه ناگهانی در آن‌ها رخ نمی‌دهد؛ مسأله‌ای که در بسیاری از پلت‌فرم‌های سخت‌افزاری که با گیت‌های متداول CMOS پیاده‌سازی می‌شوند، تحت عنوان *گلیچ*[6] شناخته می‌شود. گلیچ‌ها تغییرات ناگهانی، ناخواسته و کوتاه‌مدت سیگنال‌ها هستند که ممکن است به‌دلایل مختلفی از جمله تفاوت تأخیر المان‌های مدار، چندسطحی[7] و پیچیده‌بودن مدارها و ... ایجاد شوند و می‌توان گفت که به‌طورتقریبی اجتناب‌ناپذیر هستند. این تغییرات ناخواسته می‌توانند اطلاعات نشتی بیش از مقدار متصور از یک طرح نقاب‌گذاری معمولی را در اختیار مهاجم قرار دهند. به این ترتیب این اطلاعات اضافه‌تر، منجر به یافتن مقادیر سهم‌ها خواهند شد. در مثال زیر تأثیر گلیچ را بر یک طرح نقاب‌گذاری به‌طور دقیق‌تر می‌بینیم.

[1] Masking
[2] Secret Sharing
[3] Boolean Masking
[4] Hamming Weight
[5] Uniform
[6] Glitch
[7] Cascaded circuits

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.2.2.0]

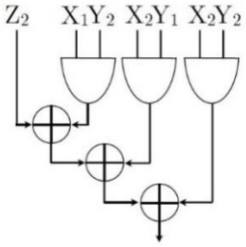
تابع رابطه (۲) را در نظر می‌گیریم:

$$f(X,Y,Z) = XY \oplus Z \quad (۲)$$

این تابع با روش نقاب‌گذاری مرتبه نخست به‌صورت رابطه (۳) پیاده‌سازی می‌شود:

$$f_1(X_1,Y_1,Z_1) = Z_1 \oplus X_1Y_1$$
$$f_2(X_2,Y_2,Z_2) = ((Z_2 \oplus X_1Y_2) \oplus X_2Y_1) \oplus X_2Y_2 \quad (۳)$$

چنان‌چه f_2 را با گیت‌های منطقی که از مدارهای CMOS ساخته می‌شوند، پیاده‌سازی کنیم، مداری مشابه شکل (۱) خواهیم داشت.



(شکل-۱): پیاده‌سازی منطقی سهم دوم از خروجی تابع $f(X, Y) = Z \veebar XY$

فرض کنیم مدار در حالت اولیه صفر است و به‌صورت تصادفی به یکی از حالت‌های دیگر می‌رود. متغیر X_2 نیز با تأخیر بیشتری نسبت به بقیه ورودی‌ها تغییر می‌کند. در این شرایط همان‌طور که در جدول (۲) می‌بینیم، بر اثر تغییر ورودی‌ها به هر یک از حالت‌های ممکن، گیت‌های AND و OR دچار گلیچ‌های متعدد و متفاوتی می‌شوند که بر مصرف توان مدار تأثیر گذاشته و توسط مهاجم قابل ردیابی هستند. به این ترتیب مهاجم می‌تواند مقدار نقاب‌گذاری‌نشده و اصلی داده Y را با استفاده از این تفاوت مصرف توان به‌دست آورد (متوسط مصرف توان در حالتی که $Y = 0$ باشد، با حالتی که $Y = 1$باشد متفاوت است).

(جدول-۲): تعداد گلیچ‌های گیت‌های AND و XOR، ناشی از تأخیر ورودی X2

Y	$Y1$	$Y2$	$X2$	$Z_2 \oplus X_1Y_2$	AND	XOR
0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	0	0	1	0	0	0
0	1	1	1	0	2	1
0	0	0	0	1	0	2
0	1	1	0	1	0	2
0	0	0	1	1	0	2
0	1	1	1	1	2	3
1	0	1	0	0	0	0
1	1	0	0	0	0	0
1	0	1	1	0	1	1
1	1	0	1	0	1	2
1	0	1	0	1	1	2
1	1	0	0	1	1	2
1	0	1	1	1	1	3
1	1	0	1	1	1	4

۴- پیاده‌سازی آستانه‌ای

روش *پیاده‌سازی آستانه‌ای*[1] نخستین‌بار در سال ۲۰۰۶، به‌منظور رفع چالش روش نقاب‌گذاری در پیاده‌سازی‌های سخت‌افزاری، ارائه شد [18] [19]. پس از آن در سال ۲۰۱۴، نسخهٔ اولیهٔ آن که امنیت مرتبهٔ نخست را برآورده می‌ساخت، به امنیت‌های مرتبهٔ بالاتر نیز تعمیم پیدا کرد [23]. این روش تاکنون بر روی اولیه‌های رمزنگاری مختلفی از جمله AES [24]، Keccak [25] و ... اجرا شده است و بهبود و بهینه‌سازی پیاده‌سازی این اولیه‌ها با استفاده از روش پیاده‌سازی آستانه‌ای همچنان نیز به‌عنوان یکی از موضوعات مهم مورد پژوهش و بررسی شناخته می‌شود. این طرح درواقع نسخهٔ اصلاح‌شدهٔ طرح‌های نقاب‌گذاری است که در سخت‌افزارها و در حضور گلیچ‌ها نیز، امنیت اثبات‌پذیر ارائه می‌کند. مزیت‌های این طرح را می‌توان نسبت به سایر روش‌های پیشنهادی در سه حوزه برشمرد [20]. نخست آن‌که برخلاف سایر راه‌کارهایی که ارائه شده‌اند، امنیت اثبات‌پذیر ایجاد می‌کند. به این معنا که از نظر ریاضی و براساس اصول تسهیم راز، می‌توان ثابت کرد که یک طرح پیاده‌سازی آستانه‌ای از مرتبهٔ dام، در برابر حملات تا مرتبهٔ dام امنیت دارد.. دوم آن‌که قابل اعمال به طیف وسیعی از پلت‌فرم‌ها، از جمله مدارات شبه CMOS[2] است که در برخی از طرح‌های پیشنهادی دیگر، با مشکل مواجه بودند. مزیت سوم آن‌که در مقایسه با سایر روش‌ها، کم‌ترین میزان منابع[3] را به ما تحمیل می‌کند.

از آن‌جا که روش پیاده‌سازی آستانه‌ای، نسخهٔ ارتقایافتهٔ روش نقاب‌گذاری است که به‌منظور به‌کارگیری در سخت‌افزارها مناسب‌سازی شده مبتنی بر همان مفاهیم تسهیم راز و MPC [4] بنا شده است و چهار ویژگی کلی دارد که دو ویژگی نخست میان تمام طرح‌های نقاب‌گذاری مشترک هستند، اما ویژگی‌های سوم و چهارم تنها مختص روش آستانه‌ای هستند. در ادامه این ویژگی‌ها به‌ترتیب توضیح داده می‌شوند.

[1] Threshold Implementation
[2] CMOS-like
[3] Resources
[4] Multi-Party Computation

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.2.2.0]

۴-۱- ویژگی نخست: یک‌نواختی نقاب‌گذاری

به این معنا که تمام حالت‌های مجازی که بتوان برای تسهیم یک متغیر به S سهم متصور شد، احتمال رخ‌دادن یکسانی داشته باشند و نتوان میان آن‌ها تمایزی قائل شد.

این ویژگی تضمین می‌کند درصورتی‌که متغیر X به s سهم تقسیم‌بندی شود، مقدار X از هر ترکیب $s-1$ تایی از سهم‌ها مستقل خواهد بود [20].

۴-۲- ویژگی دوم: صحت

تمام پردازش‌هایی که در حالت بدون نقاب، بر روی خود متغیر X اتفاق می‌افتاد تا خروجی حاصل شود، در این حالت بر سهم‌های آن اعمال می‌شوند و به جای یک خروجی، سهم‌های خروجی را خواهیم داشت. این پردازش‌ها دو گونه هستند، یا خطی و یا غیر خطی هستند. در توابع خطی وجود نقاب‌گذاری چالشی به‌دنبال نخواهد داشت، زیرا با توجه به خواص توابع خطی، همواره داریم:

$$\begin{aligned} \boldsymbol{L_{in}(X)} &= \boldsymbol{L_{in}(X_1 \oplus X_2)} \\ &= \boldsymbol{L_{in}(X_1)} \\ &\oplus \boldsymbol{L_{in}(X_2)} \end{aligned} \quad (۴)$$

اما چالش اصلی در مورد توابع غیرخطی نظیر جعبه‌های جانشانی[1] یا ... است که رابطهٔ بالا در مورد آن‌ها صادق نیست؛ یعنی:

$$\begin{aligned} \boldsymbol{S_{in}(X)} &= \boldsymbol{S_{in}(X_1 \oplus X_2)} \\ &\neq \boldsymbol{S_{in}(X_1)} \\ &\oplus \boldsymbol{S_{in}(X_2)} \end{aligned} \quad (۵)$$

بنابراین لازم است که توابع پردازش‌گر را به‌نوعی بازتعریف کنیم که همان خروجی مطلوب را در حالت نقاب‌گذاری‌شده نیز به ما بدهند. به این معنا که مجموع نتیجه‌های حاصل از پردازش سهم‌های یک متغیر یا به‌عبارت دقیق‌تر؛ مجموع سهم‌های خروجی، برابر با همان خروجی باشد که از پردازش خود آن متغیر به‌دست می‌آمد.

به‌عنوان مثال تابع رابطه (۲) را در نظر بگیرید. در رابطه (۳) پیاده‌سازی مرتبه نخست تابع با روش نقاب‌گذاری تصریح شده که در آن تعداد سهم‌های متغیرهای ورودی برابر با دو است. ملاحظه می‌شود که دو تابع پردازش‌گر f_1 و f_2 به‌گونه‌ای درنظرگرفته شده‌اند که با پردازش سهم‌های ورودی‌ها، درمجموع به‌طوردقیق همان حاصلی را به‌دست دهند که تابع f با پردازش متغیرهای اصلی می‌داد؛ بنابراین در این مثال تعداد سهم‌های ورودی برابر با دو سهم و تعداد سهم‌های خروجی نیز برابر با دو سهم شده است.

۴-۳- ویژگی سوم: Non-Completeness

براساس این ویژگی، هر کدام از توابعی که سهم‌ها را پردازش می‌کنند، لازم است که دست‌کم یکی از سهم‌های هر متغیر را به‌عنوان ورودی نداشته باشند. به این ترتیب، مهاجمی که به مقادیر این توابع دسترسی پیدا می‌کند، نخواهد توانست هیچ یک از متغیرها را بازسازی کند، چرا که از هر کدام دست‌کم یک سهم را در اختیار ندارد.

به‌طورکلی در مرتبه dام، هر ترکیب d تایی از توابع پردازش‌گر، لازم است دست‌کم یک سهم از هر کدام از متغیرها را نداشته باشند، به این ترتیب مهاجمی که بتواند از هر طریقی به d مقدار میانی دسترسی پیدا کند، با ترکیب و تحلیل اطلاعات این توابع، قادر به بازسازی هیچ یک از متغیرها نخواهد بود [20]، چرا که براساس ویژگی یک‌نواختی، هر کدام از متغیرها از هر ترکیب $s-1$ تایی از سهم‌های‌شان مستقل هستند.

واضح است که برای برقراری این ویژگی به تعداد معینی از سهم‌های خروجی و ورودی نیازمند خواهیم بود و نمی‌توان هر تعدادی دل‌خواهی برگزید. ایده‌های مختلفی پیشنهاد و آزمایش شده‌اند که بتوان به‌وسیلهٔ آن‌ها تعداد سهم‌ها را به‌نحوی تنظیم کرد که پیاده‌سازی‌های توابع ویژگی non-complete بودن را برآورده سازند. به‌عنوان مثال یکی از روش‌های پیشنهادی این است که توابع را با کم‌ترین سهم‌های متغیرهای ورودی و خروجی تنظیم کنیم و پس از آن درصورتی‌که ویژگی سوم برقرار نبود، تعداد سهم‌های ورودی یا خروجی را تا جایی که ویژگی برقرار می‌شود، افزایش دهیم. واضح‌ترین اشکال این روش سوژه محوربودن آن است. به‌عبارت دیگر، نمی‌توان یک قاعده کلی که برای همه توابع صادق باشد از آن استخراج کرد؛ بنابراین لازم بود روشی قاعده‌مند مشخص شود که با استفاده از آن بتوان توابع غیرخطی را با روشی واحد به‌صورتی‌که ویژگی سوم را برآورده سازند، پیاده‌سازی کرد. روش *تسهیم مستقیم*[2] به همین منظور ارائه شد [21]. این روش که در مرتبه‌های مختلف قابل اجرا است، به این ترتیب است که برای یک تابع غیرخطی درجه دو، توصیف منطقی هر یک از توابع پردازش‌گر یعنی A_i ها را به این شکل قرار می‌دهیم که جملات خطی آن‌ها دارای اندیس

[1] Substitution Box

[2] Direct Sharing

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

[DOR: 20.1001.1.24763047.1398.8.2.2.0]

$i+1$و جملات درجه دوم آن‌ها دارای اندیس‌های $(i+1,i+1)$، $(i+1,i+2)$ و $(i+2,i+1)$ باشند. رابطه (۶) مثالی برای پیاده‌سازی با روش تسهیم مستقیم را نشان می‌دهد:

$$
\begin{aligned}
A_1 &= 1 \oplus X_2 \oplus (X_2Y_2 \oplus X_2Y_3 \oplus X_3Y_2) \oplus (X_2Z_2 \oplus X_2Z_3 \oplus X_3Z_2) \oplus (Y_2Z_2 \oplus Y_2Z_3 \oplus Y_3Z_2)\\
A_2 &= X_3 \oplus (X_3Y_3 \oplus X_3Y_1 \oplus X_1Y_3) \oplus (X_3Z_3 \oplus X_3Z_1 \oplus X_1Z_3) \oplus (Y_3Z_3 \oplus Y_3Z_1 \oplus Y_1Z_3)\\
A_3 &= X_1 \oplus (X_1Y_1 \oplus X_1Y_2 \oplus X_2Y_1) \oplus (X_1Z_1 \oplus X_1Z_2 \oplus X_2Z_1) \oplus (Y_1Z_1 \oplus Y_1Z_2 \oplus Y_2Z_1)
\end{aligned}
\qquad (۶)
$$

این قاعدهٔ پیاده‌سازی به همین‌ترتیب قابل تعمیم به درجات بالاتر نیز هست.

با به‌کارگیری این روش دیگر نیاز نخواهد بود که تعداد سهم‌های ورودی و خروجی را به‌صورت دستی تنظیم کنیم؛ بلکه به تعداد مورد نیاز برای تکمیل قاعده تسهیم مستقیم اکتفا می‌کنیم.

نکته‌ای که در این‌جا مشهود است، افزایش تعداد سهم‌ها تحت تأثیر درجه توابع تغییر می‌کند و مطابق با یک قاعده کلی، درصورتی‌که تعداد سهم‌های ورودی و خروجی در روابط زیر صادق باشند، ویژگی سوم برقرار خواهد بود [20]:

$$S_{in} \geq t \times d + 1 \qquad (۷)$$

$$S_{out} \geq \binom{S_{in}}{t} \qquad (۸)$$

در روابط بالا t برابر با درجه تابع مورد پیاده‌سازی وd برابر با مرتبه پیاده‌سازی است.

گفتنی است که ممکن است، پیاده‌سازی‌های دیگری نیز وجود داشته باشند که با تعداد سهم‌های کمتری ویژگی سوم را برآورده سازند؛ اما بازه بالا تضمین می‌کند که ویژگی سوم به‌حتم برقرار باشد.

۴-۴- ویژگی چهارم: یک‌نواختی خروجی توابع پردازش‌گر

در پیاده‌سازی مدارها، در بسیاری از موارد با حالتی مواجه هستیم که خروجی بخشی از مدار به‌عنوان ورودی بخشی دیگر مورد استفاده قرار می‌گیرد؛ مانند خروجی توابع دور در الگوریتم‌های رمزنگاری. در چنین حالتی، درصورتی‌که بخواهیم ویژگی نخست یعنی یک‌نواختی نقاب‌گذاری، برای مدار دوم برقرار باشد، به این معنا خواهد بود که خروجی‌های مدار نخست لازم است، یکنواخت باشند.

با این حال، برقراری این ویژگی برخلاف سه ویژگی نخست با چالش‌هایی همراه است. به‌طورکلی و تاکنون، توابعی که بتوانند هر چهارویژگی را به‌صورت توأمان و قطعی برقرار سازند، پیشنهاد نشده‌اند [20]. به‌طورمعمول لازم است که برقراربودن یا نبودن ویژگی چهارم بررسی شده و درصورت عدم برقراری، با روش‌هایی جبران‌ساز آن را برآورده ساخت. تاکنون روش‌های متعددی به این منظور پیشنهاد شده‌اند که هریک مزایا و معایبی دارند.

- ***نقاب‌گذاری مجدد*** [1]

رویکرد این روش نقاب‌گذاری مجدد مقادیر میانی در تمام مراحل است [22]. این کار از طریق اضافه‌شدن مجدد مقادیر تصادفی به الگوریتم انجام می‌شود. بدیهی‌است که نخستین هزینه این روش تولید مقادیر تصادفی مازادی است که در هر مرحله مورد نیاز خواهند بود. هرچند که اصلاحاتی نیز بر روی طرح اولیه این روش انجام شد و تعداد مقادیر تصادفی مورد نیاز تا حدودی نسبت به ایده اولیه کاهش پیدا کرد، با این وجود هزینهٔ اجرایی آن هم‌چنان بالا محسوب می‌شود. البته مزیت اعمال این روش نیز در این خواهد بود که در الگوریتمی که مجهز به نقاب‌گذاری مجدد است، برقراربودن یا نبودن ویژگی نخست نیز فاقد اهمیت خواهد بود.

صورت کلی یک طرح نقاب‌گذاری مجدد به این ترتیب است که اگر فرض کنیم تابعی مانند r داشته باشیم که سهم‌های خروجی آن A_i ها باشند و تعداد آن‌ها برابر با S_{out} باشد، درصورتی‌که A_i ها یک‌نواخت نباشند، برای یک‌نواخت‌شدن‌شان کافی است $S_{out}-1$ مقدار تصادفی M_iرا تولید کرده و به توابع اضافه کنیم. رابطه (۹) فرآیند اضافه‌شدن را نشان می‌دهد:

[1] Re-Masking

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.2.2.0]

$$\mathbf{B_1} = r_1(A_1, M_1) = A_1 \oplus \mathbf{M_1}$$
$$\mathbf{B_2} = r_2(A_2, M_2) = A_2 \oplus \mathbf{M_2}$$
...
$$\begin{aligned} &B_{Sout-1} \\ &= r_{Sout-1}(A_{Sout-1}, M_{Sout-1}) \\ &= A_{Sout-1} \oplus M_{Sout-1} \end{aligned} \quad (۹)$$
$$\begin{aligned} &B_{Sout} \\ &= r_{Sout}(A_{Sout}, \mathbf{M_1}, \mathbf{M_2}, \dots M_{Sout-1}) \\ &= Asout \oplus\oplus Mi \end{aligned}$$

B_i های حاصل‌شده یک‌نواخت هستند و می‌توانند به‌عنوان ورودی تابع بعدی مورد استفاده قرار گیرند.

• افزایش تعداد سهم‌های متغیرهای ورودی

در روابط (۷ و ۸)، گفته شد که تعداد سهم‌های ورودی و خروجی، لازم است از مقدار معینی بیشتر باشند. درحقیقت آن مقادیر معین، کمینه تعداد سهم‌ها به منظور برقراری ویژگی Non-completeness هستند؛ بنابراین افزایش تعداد سهم‌ها نیز در برقراربودن آن ویژگی اختلالی ایجاد نمی‌کند. افزایش سهم‌های ورودی آنتروپی سیستم را بالا برده و به یک‌نواخت‌شدن خروجی منجر می‌شود [18]. هزینه اجرایی این روش نیز افزایش تعداد سهم‌ها و درنتیجه افزایش میزان پردازش‌های لازم و کاهش کارایی خواهد بود.

به‌عنوان مثال تابع رابطه (۱۰) را در نظر می‌گیریم که در رابطه (۱۱) با روش تسهیم مستقیم پیاده‌سازی شده است:

$$A = f(X, Y) = XY \quad (۱۰)$$

$$\begin{aligned} \mathrm{A}_1 &= f_1(X_2, X_3, Y_2, Y_3) \\ &= X_2Y_2 \oplus X_2Y_3 \\ &\oplus X_3Y_2 \end{aligned}$$
$$\begin{aligned} \mathrm{A}_2 &= f_2(X_1, X_3, Y_1, Y_3) \\ &= X_3Y_3 \oplus X_1Y_3 \\ &\oplus X_3Y_1 \end{aligned} \quad (۱۱)$$
$$\begin{aligned} \mathrm{A}_3 &= f_3(X_1, X_2, Y_1, Y_2) \\ &= X_1Y_1 \oplus X_1Y_2 \\ &\oplus X_2Y_1 \end{aligned}$$

با بررسی، مشخص می‌شود که پیاده‌سازی بالا در خروجی‌ها یک‌نواخت نیست؛ بنابراین سعی می‌کنیم با اعمال روش افزایش تعداد سهم‌های ورودی‌ها، آن را یک‌نواخت سازیم. سهم‌های ورودی‌ها را از سه به چهار افزایش می‌دهیم و خروجی‌ها را نیز به‌نحوی مدیریت می‌کنیم که ویژگی صحت برقرار باشد. در این‌جا سهم‌های خروجی را نیز از سه به چهار افزایش داده‌ایم. نتیجه به‌شکل رابطه (۱۲) خواهد بود:

$$\begin{aligned} A_1 &= (X_3 \oplus X_4)(Y_2 \oplus Y_3) \\ &\oplus Y_2 \oplus Y_3 \oplus Y_4 \\ &\oplus X_2 \oplus X_3 \oplus X_4 \end{aligned}$$
$$\begin{aligned} A_2 &= (X_1 \oplus X_3)(Y_1 \oplus Y_4) \\ &\oplus Y_1 \oplus Y_3 \oplus Y_4 \\ &\oplus X_1 \oplus X_3 \oplus X_4 \end{aligned} \quad (۱۲)$$
$$\begin{aligned} A_3 &= (X_2 \oplus X_4)(Y_1 \oplus Y_4) \oplus Y_2 \\ &\oplus X_2 \end{aligned}$$
$$\begin{aligned} A_4 &= (X_1 \oplus X_2)(Y_2 \oplus Y_3) \oplus Y_1 \\ &\oplus X_1 \end{aligned}$$

پیاده‌سازی به‌دست‌آمده در خروجی نیز یک‌نواخت است.

• کاهش تعداد سهم‌های توابع پردازش‌گر

در مواردی که تعداد سهم‌های خروجی، بیش از تعداد سهم‌های ورودی است، می‌توان با روش‌هایی هم‌چون ترکیب‌کردن سهم‌های خروجی به‌وسیلهٔ توابع آفینی[1]، تعداد آن‌ها را کاهش داد و درنتیجه ویژگی یک‌نواختی را در مورد آن‌ها برقرار ساخت [20]. با این وجود کاهش تعداد سهم‌های خروجی نمی‌تواند از مرتبه معینی بالاتر رود؛ چون در این صورت احتمال نقض‌شدن ویژگی سوم و یا نشت اطلاعات وجود دارد؛ بنابراین می‌توان گفت که این روش تنها در مواردی محدود قابل اعمال است. برای مثال تابع رابطه (۲) را در نظر می‌گیریم که در پیاده‌سازی رابطه (۱۳) خروجی‌های یک‌نواختی ندارد:

$$\begin{aligned} A_1 &= X_1Y_1 \oplus Z_1 \\ A_2 &= X_1Y_2 \\ A_3 &= X_2Y_1 \oplus Z_2 \\ A_4 &= X_2Y_2 \end{aligned} \quad (۱۳)$$

فرض کنیم A_i ها را به‌شکل رابطه (۱۴) با هم ترکیب کنیم تا تعداد سهم‌های خروجی از چهار به دو کاهش یابد:

$$\begin{aligned} B_1 &= A_1 \oplus A_3 \\ B_2 &= A_2 \oplus A_4 \end{aligned} \quad (۱۴)$$

در این حالت سهم B_2 نشت اطلاعات خواهد داشت؛ چرا که مقدار آن مطابق با رابطه (۱۵) برابر با XY_2 می‌شود که مقدار بدون نقاب و اصلی متغیر X را آشکار ساخته است.

$$\mathbf{XY_2} = X_1Y_2 \oplus X_2Y_2 \quad (۱۵)$$

[1] Affine Function

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.2.2.0]

بنابراین همان‌طور که در مثال بالا می‌بینیم، روش کاهش سهم‌های خروجی همواره کارآمد نیست و لازم است که با لحاظ دقت‌های امنیتی کافی از آن استفاده شود.

• استفاده از جملات اصلاحی[1]

جملاتی از جنس متغیرهای موجود در یک تابع، که در رابطهٔ منطقی توصیف‌کننده آن وجود ندارند؛ اما با اضافه‌شدن‌شان می‌توانند به یک‌نواخت‌شدن خروجی کمک کنند [18]. به‌عبارت دیگر، می‌توان این جملات را به‌گونه‌ای که تأثیر یکدیگر را خنثی کرده و رابطه اصلی تابع را تغییر ندهند به پیاده‌سازی توابع اضافه و ویژگی یک‌نواختی را در آن‌ها برقرار کرد. چالش اصلی این روش وجود یا عدم وجود چنین جملاتی برای هر دسته از توابع غیرخطی و چگونگی یافتن روش‌مند آن‌ها در صورت وجود داشتن است.

به‌عنوان مثال؛ تابع رابطه (۱۶) را بررسی می‌کنیم که در پیاده‌سازی تصریح‌شده در رابطه (۱۷)، ویژگی چهارم در مورد خروجی‌های آن برقرار نیست.

$$\mathbf{A} = \mathbf{f(X,Y,Z)} = \mathbf{1} \oplus \mathbf{X} \oplus \mathbf{XY} \oplus \mathbf{XZ} \oplus \mathbf{Y\,Z} \tag{۱۶}$$

$$\begin{aligned}
\boldsymbol{A_1} &= \mathbf{1} \oplus \boldsymbol{X_2} \oplus (\boldsymbol{X_2Y_2} \oplus \boldsymbol{X_2Y_3} \oplus \boldsymbol{X_3Y_2}) \oplus (\boldsymbol{X_2Z_2} \oplus \boldsymbol{X_2Z_3} \oplus \boldsymbol{X_3Z_2}) \oplus (\boldsymbol{Y_2Z_2} \oplus \boldsymbol{Y_2Z_3} \oplus \boldsymbol{Y_3Z_2}) \\
\boldsymbol{A_2} &= \boldsymbol{X_3} \oplus (\boldsymbol{X_3Y_3} \oplus \boldsymbol{X_3Y_1} \oplus \boldsymbol{X_1Y_3}) \oplus (\boldsymbol{X_3Z_3} \oplus \boldsymbol{X_3Z_1} \oplus \boldsymbol{X_1Z_3}) \oplus (\boldsymbol{Y_3Z_3} \oplus \boldsymbol{Y_3Z_1} \oplus \boldsymbol{Y_1Z_3}) \\
\boldsymbol{A_3} &= \boldsymbol{X_1} \oplus (\boldsymbol{X_1Y_1} \oplus \boldsymbol{X_1Y_2} \oplus \boldsymbol{X_2Y_1}) \oplus (\boldsymbol{X_1Z_1} \oplus \boldsymbol{X_1Z_2} \oplus \boldsymbol{X_2Z_1}) \oplus (\boldsymbol{Y_1Z_1} \oplus \boldsymbol{Y_1Z_2} \oplus \boldsymbol{Y_2Z_1})
\end{aligned} \tag{۱۷}$$

با افزودن چند جمله اصلاحی به A_i ها به رابطه (۱۸) می‌رسیم که ویژگی یک‌نواختی خروجی‌ها در آن برقرار است.

$$\begin{aligned}
\boldsymbol{B_1} &= \boldsymbol{A_1} \oplus \boldsymbol{X_2} \oplus \boldsymbol{X_3} \\
\boldsymbol{B_2} &= \boldsymbol{A_2} \oplus \boldsymbol{X_3} \oplus \boldsymbol{X_1} \\
\boldsymbol{B_3} &= \boldsymbol{A_3} \oplus \boldsymbol{X_1} \oplus \boldsymbol{X_2}
\end{aligned} \tag{۱۸}$$

می‌بینیم که جملات اصلاحی اضافه‌شده به‌نحوی هستند که در مجموع تأثیر یک‌دیگر را خنثی کرده و بر کلیت تابع تاثیری نمی‌گذارند.

• استفاده از متغیرهای مجازی[2]

گاهی می‌توان با اضافه‌کردن متغیرهایی علاوه‌بر متغیرهای اصلی خود تابع، خروجی را یکنواخت کرد [20]. با این شرط که متغیر اضافه‌شده که متغیر مجازی و سهم‌های آن که سهم‌های مجازی خوانده می‌شوند، در کلیت تابع تغییری ایجاد نکنند و برای مهاجم نیز قابل پیش‌بینی نباشند. می‌توان در مواردی این روش را معادل با نوعی پیاده‌سازی هوشمندانه از روش نقاب‌گذاری مجدد گرفت که در آن از تعداد کمتری عدد تصادفی استفاده می‌شود. با این حال، هزینه‌های تحمیلی هم‌چنان قابل صرف نظر نخواهند بود.

به‌عنوان مثال مجددا تابع رابطه (۱۰) با پیاده‌سازی غیریک‌نواخت به فرم رابطه (۱۱) در نظر می‌گیریم.

اگر بخواهیم برای یک‌نواخت‌سازی آن از روش افزودن متغیر مجازی استفاده کنیم، متغیر Z را با سهم‌های Z_1، Z_2 و Z_3 در نظر گرفته و به‌صورت رابطه (۱۹) به پیاده‌سازی اضافه می‌کنیم:

$$\begin{aligned}
\mathbf{A_1} &= \boldsymbol{f_1}(\boldsymbol{X_2}, \boldsymbol{X_3}, \boldsymbol{Y_2}, \boldsymbol{Y_3}) \\
&= \boldsymbol{X_2Y_2} \oplus \boldsymbol{X_2Y_3} \oplus \boldsymbol{X_3Y_2} \oplus \boldsymbol{X_2Z_2} \oplus \boldsymbol{X_3Z_3} \oplus \boldsymbol{Y_2\,Z_2} \oplus \boldsymbol{Y_3\,Z_3} \\
\mathbf{A_2} &= \boldsymbol{f_2}(\boldsymbol{X_1}, \boldsymbol{X_3}, \boldsymbol{Y_1}, \boldsymbol{Y_3}) \\
&= \boldsymbol{X_3Y_3} \oplus \boldsymbol{X_1Y_3} \oplus \boldsymbol{X_3Y_1} \oplus \boldsymbol{X_3Z_3} \oplus \boldsymbol{X_1Z_1} \oplus \boldsymbol{Y_3\,Z_3} \oplus \boldsymbol{Y_1\,Z_1} \\
\mathbf{A_3} &= \boldsymbol{f_3}(\boldsymbol{X_1}, \boldsymbol{X_2}, \boldsymbol{Y_1}, \boldsymbol{Y_2}) \\
&= \boldsymbol{X_1Y_1} \oplus \boldsymbol{X_1Y_2} \oplus \boldsymbol{X_2Y_1} \oplus \boldsymbol{X_1Z_1} \oplus \boldsymbol{X_2Z_2} \oplus \boldsymbol{Y_1\,Z_1} \oplus \boldsymbol{Y_2\,Z_2}
\end{aligned} \tag{۱۹}$$

[1] Correction Terms

[2] Virtual Variable

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.2.2.0]

می‌بینیم که جملات اضافه‌شده به‌واسطهٔ متغیر مجازی و سهم‌های آن، در مجموع تابع اصلی را تغییر نداده، اما در عین حال ویژگی چهارم را در آن ایجاد کرده‌اند.

• تغییر تعداد سهم‌های ورودی

در روش افزایش تعداد سهم‌های متغیرهای ورودی، سهم‌های متغیرهای خروجی نیز به تناسب افزایش می‌یابند که ویژگی صحت برقرار بماند؛ اما در برخی از پیاده‌سازی‌های مرتبه نخست، توابعی وجود دارند که می‌توان بدون نیاز به تغییردادن تعداد سهم‌های خروجی آن‌ها، تعداد سهم‌های ورودی را افزایش داد و در عین این‌که ویژگی چهارم برقرار می‌شود، ویژگی‌های قبلی نیز هم‌چنان برقرار بمانند. این روش در جلوگیری از افزایش هزینه‌های ناشی از افزایش تعداد سهم‌ها بسیار مؤثر است؛ اما همان‌طور که مشخص است، تنها در موارد محدودی قابل اعمال است [20].

برای مثال همان تابع رابطه (۱۰) را در نظر می‌گیریم که پیاده‌سازی غیریکنواخت آن با روش تسهیم مستقیم دارای سه سهم ورودی و سه سهم خروجی بود. در روش افزایش تعداد سهم‌های ورودی‌ها که در رابطه (۱۲) بررسی شد، ملاحظه شد که می‌توان با افزایش سهم‌های ورودی و هم‌چنین خروجی از سه به چهار، یک تابع یک‌نواخت ساخت؛ اما همین تابع را می‌توان با افزایش سهم‌های ورودی از سه به چهار و ثابت‌نگه‌داشتن سهم‌های خروجی در همان سه سهم نیز، به‌صورت یک‌نواخت پیاده‌سازی کرد. رابطه (۲۰) این پیاده‌سازی را نشان می‌دهد:

$$\begin{aligned} A_1 &= (X_2 \oplus X_3 \oplus X_4)(Y_2 \\ &\quad \oplus Y_3) \oplus Y_4 \\ A_2 &= (X_1 \oplus X_3)(Y_1 \oplus Y_4) \\ &\quad \oplus X_1Y_3 \oplus X_4 \\ A_3 &= (X_2 \oplus X_4)(Y_1 \oplus Y_4) \\ &\quad \oplus X_1Y_2 \oplus X_4 \\ &\quad \oplus Y_4 \end{aligned} \qquad (۲۰)$$

• تجزیه توابع درجه بالا به درجات پایین‌تر

در مواردی که توابع غیرخطی مانند جعبه جانشانی، با درجات بالا داریم، می‌توانیم با تجزیه این توابع به توابعی با درجات پایین‌تر که یا خود یک‌نواخت هستند یا ویژگی یک‌نواختی راحت‌تر در موردشان برقرار می‌شود، پیچیدگی مسأله را کاهش دهیم [20]. برای مثال می‌توان یک تابع غیر خطی با درجه جبری سه را به دو تابع غیرخطی با درجه جبری دو تجزیه کرد.

$$sbox(X) = f(g(X)) \qquad (۲۱)$$

۵- جمع‌بندی

پیاده‌سازی‌های متداولی که برای الگوریتم‌های امن در مدل جعبه‌سیاه وجود دارند، اغلب در مدل حمله جعبهٔ خاکستری دارای نشت اطلاعات هستند. این نشت اطلاعات معلول روش‌های پیاده‌سازی و ابزارهای فیزیکی مورد استفاده است و درصورت عدم کنترل می‌توانند تا دست‌یافتن به کلید مخفی الگوریتم‌ها برای مهاجم سودمند باشد؛ لذا ضروری است، روش‌هایی ابداع و اعمال شوند که بتوان به‌کمک آن‌ها این جریان‌های نشت اطلاعات را مدیریت کرد.

در همین راستا، روش پیاده‌سازی آستانه‌ای مبتنی بر روش کلی نقاب‌گذاری، به‌منظور برقراری امنیت در سخت‌افزارها که با مسائلی هم‌چون گلیچ‌ها و ... مواجه هستند، ارائه شده است و امنیتی قابل اثبات فراهم می‌کند. با این حال خود این روش نیز هم‌چنان با چالش‌های مواجه است که افق مدنظر در تأمین امنیت سخت‌افزاری را که همانا برقراری امنیت در برابر مهاجم با هر مرتبه‌ای از توان است، با سؤال مواجه می‌کنند. نخستین طرح‌های پیشنهادی این روش، پیاده‌سازی های آستانه‌ای مرتبه دوم بودند که امنیت را در برابر مهاجمان با توان مرتبه یک، ایجاد می‌کردند؛ اما به‌مرور ایده‌های مقابله با مهاجمان مرتبه بالاتر نیز مطرح و پرداخته شدند که در این مقاله ویژگی‌های مورد نیاز برای نیل به این منظور توضیح داده شد؛ اما هم‌چنان پرسش‌هایی نظیر مدیریت بده‌بستان‌های ابعادِ امنیتِ اعداد تصادفی مطرح هستند که لازم است در مورد آن‌ها بررسی‌های بیشتری به‌عمل آید. نتایجی که تاکنون به‌دست آمده‌اند، نشان می‌دهند که به‌الزام ارتباط مستقیمی میان کاهش تعداد سهم‌ها و به تبع کاهش اعداد تصادفی مورد نیاز، با کاهش ابعاد وجود ندارد و یا آن‌که هرچند افزایش سهم‌ها منجر به افزایش امنیت خواهد شد، اما افزایش سهم‌ها از یک میزان به بعد، افزایش چشم‌گیر ابعاد و منابع مورد نیاز را رقم می‌زند که در مواردی به هیچ وجه مقرون به صرفه نخواهد بود.

از دیگر پرسش‌های موجود، چالش‌های ناظر به ویژگی‌های چهارگانه پیاده‌سازی آستانه‌ای است؛ نظیر یافتن روش‌هایی بهینه برای یک‌نواخت‌سازی خروجی‌ها و...

[Downloaded from monadi.isc.org.ir on 2026-11-20]
[DOR: 20.1001.1.24763047.1398.8.2.2.0]

هم‌چنین با توجه به اهمیت‌یافتن برقرای امنیت در پیاده‌سازی‌هایی که امروزه با سرعتی بسیار زیاد به سمت کاهش ابعاد و کاهش مصرف توان پیش می‌روند، مفید خواهد بود که از مدل‌های نشت اطلاعات دقیق‌تری استفاده شود که در آن‌ها تأثیرات المان‌های مدار بر یک‌دیگر، نشتی‌های ناشی از این تأثیرات و ... نیز درنظر گرفته شوند.

۶- مراجع

[1] H. M. Heys, "A tutorial on linear and differential cryptanalysis," vol. 26, no. 3, 2002.

[2] S. Chow, P. Eisen, H. Johnson, & P.C. Van Oorschot, "White-box cryptography and an AES implementation," *International Workshop on Selected Areas in Cryptography,* pp. 250-270, 2002.

[3] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Annual International Cryptology Conference,* pp. 104-113, 1996.

[4] K. Gandolfi, C. Mourtel, and F. Olivier., "Electromagnetic analysis: Concrete results," *Cryptographic Hardware and Embedded Systems—CHES,* pp. 675, 2001.

[5] P.Kocher, J.Jaffe, & B. Jun, "Differential power analysis," *Annual International Cryptology Conference,* pp. 388-397, 1999.

[6] O. Kömmerling, & M.G.Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors," *Smartcard,* vol. 99, pp. 9-20, 1999.

[7] D.Samyde, S. Skorobogatov , R.Anderson, & J. J. Quisquater, "On a new way to read data from memory," *First International IEEE Security in Storage Workshop,* pp. 65-69, 2002.

[8] K.Pietrzak ,and S. Dziembowski, "Leakage-resilient cryptography," *Foundations of Computer Science,*pp. 293–302, 2008.

[9] M. Medwed, F. Standaert, and A. Joux, "Towards super-exponential side-channel security with efficient leakage-resilient prfs," *Cryptographic Hardware and Embedded Systems - CHES,* vol. 7428 of LNCS, pp. 193–212, 2012.

[10] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," *IEEE Computer Society,* 2004.

[11] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert, "Shuffling against side-channel attacks: A comprehensive study with cautionary note," *Advances in Cryptology – ASIACRYPT,* vol. 7658 of LNCS, pp. 740–757, 2012.

[12] F. Durvaux, M. Renauld, F.-X. Standaert, L. van Oldeneel tot Oldenzeel, and N. Veyrat-Charvillon, "Random delay countermeasure," *Cryptanalysis of the CHES 2009/2010.*

[13] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," *Advances in Cryptology - CRYPTO,* vol. 2729 of LNCS, pp. 463–481, 2003.

[14] L. Goubin and J. Patarin, "DES and differential power analysis the "duplication" method," *Cryptographic Hardware and Embedded Systems,* vol. 1717 of LNCS, pp. 158–172, 1999.

[15] S. Chari, C. Jutla, J. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," *Advances in Cryptology — CRYPTO,* vol. 1666 of LNCS, pp. 398–412, 1999.

[16] T. S. Messerges, "Securing the AES finalists against power analysis attacks," *Fast Software Encryption,* vol. 1978 of LNCS, pp. 150–164, 2001.

[17] E. Trichina, T. Korkishko, and K. Lee, "Small size, low power, side channelimmune AES coprocessor: Design and synthesis results," *Advanced Encryption Standard – AES,* vol. 3373 of LNCS, pp. 113–127, 2005.

[18] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," *Information and Communications Security,* vol. 4307 of LNCS, pp. 529–545, 2006.

[19] E. Prouff and T. Roche, "Higher-order glitches free implementation of the AES using secure multi-party computation protocols," *Cryptographic Hardware and Embedded Systems - CHES,* vol. 6917 of LNCS, pp. 63–78, 2011.

[20] B. Bilgin, "Threshold implementations: as countermeasure against higher-order differential power analysis," 2015.

[21] S. Nikova, V. Rijmen, and M. Schläffer, "Secure hardware implementation of nonlinear functions in the presence of glitches," *Cryptology,* vol. 24, no. 2, pp. 292–321, 2011.

[22] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of AES," *EUROCRYPT,* vol. 6632 of LNCS, pp. 69–88, 2011.

[23] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "Higher-order threshold implementations." In International Con-

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

[DOR: 20.1001.1.24763047.1398.8.2.2.0]

ference on the Theory and Application of Cryptology and Information Security, pp. 326-343. Springer, Berlin, Heidelberg, 2014.

[24] T. Sugawara,"3-share threshold implementation of AES S-box without fresh randomness." *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019, pp.123-145.

[25] J. Daemen, "Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing." In International Conference on Cryptographic Hardware and Embedded Systems, pp. 137-153. Springer, Cham, 2017.



سارا زارعی کارشناسی الکترونیک خود را در سال ۱۳۹۶ از دانشگاه تهران دریافت کرد و در سال ۱۳۹۹، از مقطع کارشناسی ارشد در رشتهٔ مخابرات امن و رمزنگاری از دانشگاه شهید بهشتی فارغ‌التحصیل شد. ازجمله زمینه‌های پژوهشی مورد علاقه او می‌توان به حملات فیزیکی، پیاده‌سازی امن، تحلیل و ارزیابی امنیت سخت‌افزارها و نیز مباحث مربوط به زنجیره بلوکی اشاره نمود.



هادی سلیمانی در سال ۱۳۸۷ مدرک کارشناسی مخابرات را از دانشگاه علم و صنعت ایران، کارشناسی‌ارشد را از دانشگاه امام حسین (ع) با گرایش مخابرات رمز در سال ۱۳۸۹ و سپس مدرک دکترا را در دانشکده علوم کامپیوتر دانشگاه آلتو فنلاند درسال ۱۳۹۴ اخذ کرد. همچنین طی دو دوره کوتاه مدت پسادکترا در گروه رمزنگاری دانشگاه DTU دانمارک درخصوص تحلیل و طراحی رمزهای قالبی نوین مشغول به پژوهش و شد. وی هم‌اکنون، ضمن همکاری با پژوهشکده‌ها و مراکز پژوهشی مختلف در حوزه رمزنگاری و امنیت اطلاعات، به‌عنوان استادیار گروه امنیت شبکه و رمزنگاری پژوهشکده فضای مجازی دانشگاه شهید بهشتی مشغول به کار است. موضوعات پژوهشی مورد علاقه ایشان تحلیل و طراحی اولیه‌های رمزنگاری، پیاده‌سازی امن و حملات کانال جانبی است.

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

[DOR: 20.1001.1.24763047.1398.8.2.2.0]

[DOR: 20.1001.1.24763047.1398.8.2.2.0] [Downloaded from monadi.isc.org.ir on 2026-11-20]