ارائهٔ یک حملهٔ SPA موفّق به پیاده‌سازی AES روی میکروکنترلر PIC

حامد یوسفی[1]، محمود گردشی[2] و محمّد سبزی‌نژاد فراش[3]
[1و2]دانشکدهٔ فاوای دانشگاه جامع امام حسین (ع) ، تهران، ایران
[3]دانشکدهٔ علوم ریاضی و کامپیوتر دانشگاه تربیت معلم، تهران، ایران

چکیده

در این مقاله یک حملهٔ تحلیل توانی ساده (SPA) به پیاده‌سازی AES روی میکروکنترلر PIC را معرّفی می‌کنیم. این حمله تبدیل مخلوط‌ساز ستونی (MixColumns) را از الگوریتم AES هدف قرار می‌دهد. با استفاده از اطّلاعات به‌دست‌آمده از سیگنال توان مصرفی حین پردازش داده‌ها در این تبدیل می‌توان فضای کلید را محدود کرد. این حمله می‌تواند فضای جستجوی کلید را بسته به تکرار حمله، به‌طور دقیق تا یک کلید محدود کند. اساس حمله بر مبنای تشخیص نتیجهٔ یک عمل‌گر شرطی در تبدیل مخلوط‌ساز ستونی با استفاده از اثرهای توان است.

واژگان کلیدی: حملات کانال جانبی، حملهٔ تحلیل توانی ساده، الگوریتم AES، میکروکنترلرPIC.

۱- مقدمه

تا اواسط دههٔ ۱۹۹۰، ابزارهای رمزنگاری، به‌صورت یک جعبهٔ سیاه درنظرگرفته می‌شدند. این ابزارها با گرفتن دادهٔ ورودی و با استفاده از کلید مخفی خود، خروجی را تولید می‌کردند. بنابراین حملات با استفاده از متن خوانای شناخته‌شده، متن رمزی شناخته‌شده و یا یک زوج متن خوانا و رمزی شناخته‌شده انجام می‌شد و به‌نظر نمی‌آمد که هیچ‌گونه اطّلاعات اضافه دیگری در دسترس باشد. امروزه مشخّص‌شده که این فرض صحیح نیست. در پیاده‌سازی‌های فیزیکی، همواره خروجی‌های دیگری وجود دارند که باعث نشت اطّلاعات از سامانهٔ رمزنگاری می‌شوند که به آن‌ها اطّلاعات کانال جانبی گویند. مهاجم با داشتن دسترسی به رمزکننده و با جمع‌آوری و تحلیل این داده‌ها می‌تواند سامانهٔ رمزنگاری را مورد حمله قرار دهد. به این نوع تحلیل، تحلیل کانال جانبی یا حملهٔ کانال جانبی گویند (*Goldack, 2008*).

انواع مختلفی از حملات کانال جانبی به سامانه‌های رمزنگاری ارائه شده‌اند که هدف همهٔ این‌حملات به‌دست‌آوردن کلید رمزنگاری است. حملات به سامانه‌های رمزنگاری از نظر هزینه، زمان، تجهیزات و تخصص لازم متفاوت‌اند و روش‌های گوناگونی برای تقسیم‌بندی آنها وجود دارد. در یک تقسیم‌بندی، آنها را به‌صورت فعّال و غیر فعّال درنظرمی‌گیرند (*Mangard et al, 2007, Mangard, 2004*). حملات فعّال سعی در به‌دست‌آوردن کلید با استفاده از برهم‌زدن عملکرد طبیعی سامانه دارند. در یک حملهٔ فعّال، ابزار رمزنگاری یا ورودی‌ها و یا محیط، طوری دست‌کاری می‌شود که سامانه دچار رفتاری غیر معمول شود. سپس با تحلیل این رفتار غیر معمول، سعی در بازیابی کلید می‌شود. برای مثال، می‌توان حملهٔ القای خطا[1] را نام برد که در آن با القای تعمدی خطا در محاسبات و استفاده از نتایج آن به سامانهٔ رمزنگاری حمله می‌کنند. نمونه‌ای از حملهٔ القای خطا روی *AES* در (*Chen and Yen, 2003*) ارائه شده است. در مقابل حملات فعّال، حملات غیر فعّال قرار می‌گیرند که عملکرد سامانه را در حالت عادی زیر نظر قرار می‌دهند و در عملکرد متعارف آن اختلالی ایجاد نمی‌کنند.

حملات را از جنبهٔ دیگر می‌توان به تهاجمی[2]، نیمه تهاجمی[3] و غیر تهاجمی[1] تقسیم کرد که خود می‌توانند

[1]Fault Induction Attacks
[2] Invasive
[3] Semi-Invasive

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]

فعال یا غیر فعال باشند (*Mangard et al, 2007, Mangard, 2004*). در حملات تهاجمی با برداشتن درپوش یا پوشش تراشه، سعی در به‌دست آوردن اطّلاعات از رمزکننده می‌شود. در این نوع حملات به‌طور اساسی هیچ‌گونه محدودیتی در برخورد با رمزکننده برای بازیابی کلید وجود ندارد. به‌عنوان مثال با وصل‌کردن یک سیم به گذرگاه داده[2]، می‌توان اطّلاعات مبادله شده را آشکار کرد. از این نوع حمله می‌توان حملهٔ پروب‌گذاری[3] را نام برد که در ایستگاه پروب‌گذاری و با استفاده از تجهیزات دقیق و میکروسکوپ‌های با دقّت بالا و پروب‌های بسیار نازک انجام می‌شود. با این تجهیزات می‌توان به بازیابی داده‌ها در نقاط مختلف تراشه و یا مشاهدهٔ سلول‌های حافظه پرداخت. چنین سامانه‌ای به‌طور گسترده در صنایع نیمه‌هادی مورد استفاده قرار می‌گیرد. این حملات بسیار قدرتمند بوده امّا نیازمند تجهیزات با قیمت بالا هستند. درنتیجه کارهای اندکی در این زمینه ارائه شده (*Mangard et al, 2007*) و همچنین این حملات باعث آسیب فیزیکی به سخت‌افزار رمزنگاری خواهند شد. یک مدل حملهٔ پروب‌گذاری روی *AES* در (*Schmidt and Kim, 2008*) ارائه شده است.

در حملات نیمه‌تهاجمی، پوشش تراشه برداشته می‌شود امّا هیچ‌گونه اتصال الکتریکی مستقیمی به سطح تراشه انجام نمی‌شود. به‌طور معمول در حملات نیمه‌تهاجمی غیرفعال، هدف خواندن محتوای سلول‌های حافظه بدون استفاده از پروب‌گذاری است. یک حملهٔ موفّق در این مورد در (*Samyde et al, 2002*) آمده است. در حملات نیمه‌تهاجمی فعال، هدف القای خطا به سامانه است.

این کار با اشعهٔ *X*، میدان‌های الکتریکی و مغناطیسی و یا نور انجام می‌شود. برای مثال، یک حملهٔ القای خطا به کمک نور را می‌توان در (*Skorobogatov and Anderson, 2003*) دید. حملات نیمه‌تهاجمی به تجهیزات گران‌قیمت مانند تجهیزات حملات تهاجمی نیاز ندارند؛ امّا هنوز برای انجام یک حملهٔ نیمه‌تهاجمی موفّق تلاش بالایی نیاز است. برای مثال فرآیند مشخّص‌کردن مکان دقیق حمله، روی سطح یک تراشهٔ مدرن، به تجربه و زمان کافی نیاز دارد (*Mangard et al, 2007*). در این زمینه (*Skorobogatov, 2005*) منبع کاملی است.

حملات غیرتهاجمی دستهٔ دیگر حملات بودند. در این گونه حملات بدون آسیب فیزیکی و برداشتن پوشش تراشه، سعی می‌شود سامانه شکسته شود. بیشتر حملات غیر تهاجمی با تجهیزات به‌نسبه ارزان‌تری قابل اجرا بوده و بنابراین یک تهدید جدّی برای امنیت ابزارهای رمزنگاری هستند. در سال‌های اخیر حملات غیرتهاجمی‌غیرفعال بسیار مورد توجه قرار گرفته‌اند. مهم‌ترین این حملات از اطّلاعاتی مانند زمان، توان مصرفی ابزار و یا تشعشعات الکترومغناطیسی استفاده می‌کنند که به‌ترتیب حملات زمانی[4] (*Kocher, 1996*)، حملات تحلیل توان[5] (*Kocher et al, 1999*)و حملات الکترومغناطیسی *(EM)*[6] (*Gandolfi et al, 2001,Quisquater and Samyde, 2001*) را شامل می‌شوند. حملات غیرتهاجمی‌فعّال نیز وجود دارند که هدف این حملات تزریق خطا به سامانه بدون برداشتن پوشش آن است. برای نمونه این خطا می‌تواند به‌وسیلهٔ اختلال[7] در پالس ساعت، اختلال در توان مصرفی و یا تغییر دمای محیط باشد. در (*Whelan et al, 2006*) یک جمع‌بندی از این نوع حملات آمده است.

در حملات زمانی از زمان اجرای الگوریتم و مدتی که طول می‌کشد تا سامانه به یک ورودی پاسخ دهد، برای انجام حمله استفاده می‌شود. زمان اجرای یک الگوریتم می‌تواند حاوی اطّلاعاتی دربارهٔ مقادیر مخفی در رمزنگاری باشد. به‌طور معمول برنامه‌نویسان و طراحان الگوریتم سعی در کاهش زمان اجرا و افزایش سرعت دارند. زمان اجرای الگوریتم، به‌ازای ورودی‌های مختلف ممکن است اندکی متفاوت باشد. اگر چنین باشد و زمان اجرای الگوریتم برای ورودی‌های مختلف به‌دقّت ثبت شود، آنگاه با تحلیل آماری می‌توان به جستجوی همبستگی بین اندازه‌گیری‌های مختلف زمان و بعضی بیت‌های کلید پرداخت. ایدهٔ استفاده از زمان اجرای الگوریتم ابتدا توسط کوچر[8] (*Kocher, 1996*) مطرح شد. در مقالهٔ کوچر روشی توضیح داده‌شد که در آن از اطّلاعات زمانی برای به‌دست آوردن کلید مخفی در یک پیاده‌سازی *RSA* استفاده شده بود. در (*Koeune and Quisquater, 1999*) روشی برای شکستن *AES* با تحلیل زمان آمده است. این حمله با سه‌هزار بار اندازه‌گیری زمان انجام شده است و از اختلاف زمان ایجاد شده در تابع مخلوط‌ساز ستونی[9] از الگوریتم *AES* به‌ازای ورودی‌های مختلف استفاده شده است.

ما در این مقاله رمزکننده را با استفاده از تحلیل توان مصرفی مورد حمله قرار می‌دهیم. در حملات تحلیل توان، به دنبال یافتن وابستگی در سیگنال توان مصرفی سخت‌افزار

[1] Non-Invasive
[2] Data Bus
[3] Probing Attack

[4] Timing Attacks
[5] Power Analysis Attacks
[6] Electromagnetic (EM)Attacks
[7] Glitch
[8] Paul C. Kocher
[9] Mix-Columns

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]

رمزنگاری هستیم. در حالت کلی دو نوع وابستگی در سیگنال توان وجود دارد: وابستگی به عمل‌ونـد (داده) و وابستگی به عمل‌گر. حملات تحلیل توان از این حقیقت نتیجه می‌شوند که توان مصرفی ابزار رمزنگاری وابسته به داده‌هایی است که پردازش می‌کند و عمل‌گرهایی که اجرا می‌کند. ایدهٔ استفاده از این حقیقت در تحلیل ابزارهای رمزنگاری نیز ابتدا توسط کوچر (*Kocher et al, 1999*) مطرح شد. حملات تحلیل توان را می‌توان به دو دستهٔ ساده[1] (*SPA*) و تفاضلی[2] (*DPA*) تقسیم کرد. تحلیل توان ساده تکنیکی است که شامل مشاهده و بررسی مستقیم اثرهای توان جمع‌آوری‌شده حین رمزنگاری است. به عبارت دیگر مهاجم سعی دارد با تعداد محدودی اثر توان، کم و بیش به کلید و یا اطّلاعاتی دربارهٔ الگوریتم رمزنگاری دست یابد. در *SPA* اغلب نیازمند داشتن اطّلاعاتی از جزئیات پیاده‌سازی ابزار مورد حمله هستیم. در عمل *SPA* زمانی مفید و مورد استفاده قرار می‌گیرد که یک یا تعداد اندکی اثر توان برای یک مجموعه ورودی داشته باشیم. امّا در تحلیل تفاضلی توان، هدف یافتن کلید با استفاده از تعداد زیادی اثر توان است. این سیگنال‌های توان در حین رمزنگاری و یا رمزگشایی بلوک‌های دادهٔ مختلف ذخیره شده‌اند و برای تحلیل آنها از روش‌های آماری استفاده می‌شود. در *DPA* نیازمند داشتن اطّلاعات دقیق درخصوص ابزار مورد حمله نیستیم. به‌طور معمول تنها کافی است بدانیم که الگوریتم در حال اجرا روی سخت‌افزار رمزنگار چیست. همچنین در این حمله مهاجم می‌تواند کلید را در شرایط به‌شدّت نوفه‌ای نیز کشف کند. در مقایسه با *SPA*، در *DPA* نیازمند داشتن اثرهای توان بسیار زیادی هستیم. بنابراین برای اعمال حمله لازم است که ابزار مورد حمله، مدّتی در دسترس فیزیکی باشد. نتایج زیادی از اعمال *DPA* روی الگوریتم‌های مختلف به‌خصوص *AES* گزارش شده است. در (*Masoomi et al, 2010*) نمونه‌ای از این حمله آمده است.

ما در اینجا به اعمال یک حملهٔ *SPA* موفّق روی *AES* می‌پردازیم. حملات *SPA* با تعداد اندکی اثر توان سعی در شکستن سامانه دارند. در زمینهٔ اعمال *SPA* روی *AES* کارهای اندکی صورت گرفته است. در (*Mangard, 2003*) یک نمونه از این حمله به *AES* آمده است که پیاده‌سازی تابع توسیع کلید[3] از *AES* را هدف قرارداده‌است. در این حمله از نشت اطّلاعات و وجود رابطهٔ بازگشتی در محاسبهٔ کلیدهای دور *AES* در تابع توسیع کلید جهت کاهش فضای کلید استفاده شده است. اساس حمله برمبنای تشخیص وزن همینگ نتایج میانی حین اجرای تابع توسیع کلید در الگوریتم پیاده‌سازی شده برای کارت هوشمند است. موفقیت آن مشروط به تشخیص وزن همینگ داده‌های میانی است و از آنجا که در تشخیص وزن همینگ داده‌ها از روی اثر توان عوامل زیادی تاثیر گذارند (از جمله مصرف توان عملگر و عمل‌گرهای قبل و بعد از آن، وضعیت گذرگاه داده قبل و بعد از اجرای عملگر و همچنین ساختار داخلی سخت‌افزار و ارتباط اجزای مختلف آن)، این تشخیص مشکل شده و باعث ناکارآمدی و پرهزینه شدن حمله می‌شود.

در این مقاله یک حملهٔ *SPA* موفّق روی *AES* ارائه می‌کنیم و نشان می‌دهیم که در یک پیاده‌سازی ناامن از *AES*، با استفاده از تحلیل سیگنال توان حاصل از پردازش داده‌های مختلف حین اجرای تبدیل مخلوط‌ساز ستونی، چگونه می‌توان یک حملهٔ تحلیل سادهٔ‌توان موفّق و کارآمد داشت. در اینجا از تفاوت ایجاد شده در اثر توان مصرفی رمزکننده به ازای نتایج مختلف یک عمل‌گر شرطی در تبدیل مخلوط‌ساز ستونی استفاده خواهیم کرد.

در ادامهٔ این مقاله، الگوریتم *AES* و چگونگی اجرای تبدیل مخلوط‌ساز ستونی در بخش دو خواهند آمد. در بخش سه به بیان نقطهٔ آسیب‌پذیر آن برای اعمال حمله می‌پردازیم و در ادامه روش نظری اعمال ایدهٔ حمله را در بخش چهار ارائه می‌کنیم. پیاده‌سازی عملی حمله را در بخش پنج خواهیم داشت و در بخش شش نتایج عملی حمله را می‌آوریم. در انتها نیز پس از پیشنهاد راه‌کاری برای مقابله با این تهدید به جمع‌بندی و نتیجه‌گیری می‌پردازیم.

۲- الگوریتم AES

الگوریتم رمز رایندال[4] (*Daemen and Rijmen, 2000*) توسط *NIST* در نوامبر ۲۰۰۱ به‌عنوان استاندارد پیشرفتهٔ رمزنگاری (*AES*) معرفی و سپس تحت استاندارد *FIPS197* (*NIST, 2001*) ارائه شد. *AES* یک الگوریتم رمز قالبی بایت‌گرا با کلید متقارن و طول بلوک ۱۲۸ بیت است. طول کلید می‌تواند ۱۲۸، ۱۹۲ و ۲۵۶ بیت باشد و متناسب با طول کلید، تعداد دورهای الگوریتم به‌ترتیب برابر ۱۰، ۱۲، و ۱۴ دور است. برای مشخص کردن طول کلید الگوریتم مورد بررسی، *AES* را به‌صورت *AES-128*، *AES-192* و *AES-256* نمایش می‌دهند. هر دور از *AES* شامل چهار تابع است: جانشینی بایتی[5]، شیفت سطری[6]، مخلوط‌ساز ستونی و جمع

[1] Simple Power Analysis
[2] Differential Power Analysis
[3] Key Expansion

[4] Rijndael
[5] SubBytes
[6] ShiftRows

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]

با زیرکلید دور[1]، که به‌ترتیب در هر دور اجرا می‌شوند. پیش از دور اوّل به‌منظور سفیدسازی، یک مرحله جمع با زیرکلید وجود دارد. همچنین تابع مخلوط‌ساز ستونی در آخرین دور وجود ندارد. طول بلوک ۱۲۸ بیت است و در آغاز عملیات رمزنگاری، پیام به بلوک‌های ۱۲۸ بیتی (۱۶ بایتی) تقسیم می‌شود و به‌صورت ماتریسی ۴×۴ نمایش داده می‌شود که به آن ماتریس حالت گویند. جمع با زیرکلید دور عبارت است از یک عملیات سادهٔ *XOR* که در آن عناصر ماتریس حالت با زیرکلید هر دور (*RoundKey*) بایت به بایت *XOR* می‌شوند. جانشینی بایتی یک تابع غیر خطی است که هر بایت از ماتریس حالت را با بایت متناظر آن در جدول *S-Box* جایگزین می‌کند. جدول *S-Box* شامل مقادیر معکوس ضربی همهٔ حالت‌های ممکن یک بایت (۲۵۶ حالت) در $GF(2^8)$ با یک تبدیل آفینی است. تبدیل شیفت سطری یک تابع خطی است که سطر اول را تغییر نمی‌دهد و سطرهای دوم، سوم و چهارم را به‌ترتیب یک بایت، دوبایت و سه بایت به‌صورت چرخشی به سمت چپ انتقال می‌دهد. در تبدیل مخلوط‌ساز ستونی، ضرب ماتریسی زیر روی ستون‌های ماتریس حالت اعمال می‌گردد.

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

منظور از عمل‌گر●،ضرب در فضای $GF(2^8)$ است. این ضرب‌ها به‌صورت ضرب چندجمله‌ای‌های باینری به پیمانهٔ چند جمله‌ای تحویل‌ناپذیر $m(x)=x^8+x^4+x^3+x+1$ روی $GF(2)$ تعریف می‌شوند.

۳- نقطهٔ آسیب‌پذیر

در هنگام پیاده‌سازی *AES*، تابع مخلوط‌ساز ستونی را باتوجّه به ماتریس بالا و چندجمله‌ای تحویل‌ناپذیر *m(x)*، می‌توان بسیار کارآمد پیاده‌سازی کرد (*NIST, 2001*). ماتریس ضربی فقط شامل عناصر *'01'*، *'02'* و *'03'* است و از آنجا که *'01'+'02'='03'* است، پس تنها ضربی که باید در عمل پیاده‌سازی شود همان ضرب در *'02'* است. همچنین یک بایت داده را می‌توان به‌صورت چند جمله‌ای زیر در نظر گرفت:

$$b(x) = b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b : \{b_7\,b_6\,b_5\,b_4\,b_3\,b_2\,b_1\,b_0\} \quad (۱)$$

و برای *m(x)* داریم:

$$m(x) = x^8+x^4+x^3+x+1 : \{100011011\} : \{01\}\{1B\} \quad (۲)$$

ضرب‌کردن *'02'* در بایت داده، معادل ضرب کردن *x* در *m(x)* است و عبارت است از:

$$x \bullet b(x) = b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x \quad (۳)$$

که برای رسیدن به جواب باید به پیمانه *m(x)* کاهش یابد. اگر $b_7 = 0$ باشد پاسخ به‌خودی‌خود کاهش‌یافته است. اما اگر $b_7 = 1$ باشد، آنگاه کاهش با استفاده از کم‌کردن (*XOR* کردن) چندجمله‌ای *m(x)* حاصل می‌شود. عملیات ضرب *x* در *b(x)* به پیمانه *m(x)* را *xTime* گویند. در واقع ضرب در *'02'* را می‌توان این‌چنین سرعت بخشید:

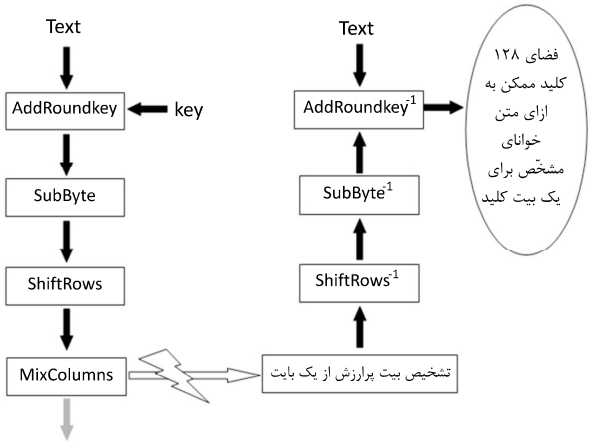
۱) بایت مورد نظر را یک واحد به چپ شیفت می‌دهیم.

۲) اگر بیت نقلی[2] ایجاد شد، نتیجه با *'1B' XOR* می‌شود.

اگر پیاده‌سازی با دقّت انجام نشود، عملیات بالا که شامل یک عمل شرطی است، می‌تواند در حالات مختلف اثر متفاوتی روی نمودار توان بگذارد و باعث نشت اطّلاعات شده و مقدار بیت پرارزش بایت ضرب شده را آشکار کند. در این مقاله از این نقطه‌ضعف برای پی‌ریزی حمله استفاده می‌کنیم.

۴- روش اعمال نظری حمله

در قسمت قبل نقطه‌ضعف موجود در تابع مخلوط‌ساز ستونی را بیان کردیم. در این قسمت روش استفاده از این نقطه‌ضعف برای اعمال حمله و بازیابی کلید را بیان می‌کنیم. اساس حمله عبارت است از تشخیص بیت پرارزش بایت‌های ماتریس حالت در تبدیل مخلوط‌ساز ستونی دور اوّل با استفاده از اثرهای توان.



(شکل ۱): الگوریتم AES تا تبدیل مخلوط‌ساز ستونی دور اول و روند بازگشت الگوریتم با داشتن بیت پرارزش یک بایت در تبدیل مخلوط‌ساز ستونی

[1] AddRoundKey

[2] Carry

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]

در سمت چپ از (شکل ۱) الگوریتم *AES* تا رسیدن به خروجی اولین تبدیل مخلوط‌ساز ستونی آمده است. حال اگر بیت پرارزش بایتی از ماتریس حالت را در تبدیل مخلوط‌ساز ستونی تشخیص داده باشیم، می‌توانیم در جهت عکس نمودار به سمت بالا حرکت کنیم تا به یک فضای کلید کاهش‌یافته برسیم.

با داشتن بیت پرارزش بایتی از ماتریس حالت، وقتی از $SubBytes^{-1}$ عبور کنیم یا نیمهٔ بالای جدول معکوس *S-Box* انتخاب می‌شود و یا نیمهٔ پایین آن که ۱۲۸ عضو ممکن از جدول برای هریک از حالات صفر یا یک بودن بیت پرارزش انتخاب می‌شوند. حال با *XOR* کردن فضای انتخابی با بایت متناظر از متن خوانا، یک فضای کلید با ۱۲۸ عضو برای یک بایت از کلید به‌دست می‌آید.

اگر عمل بالا را برای ورودی دیگری تکرار کنیم و بیت پرارزش از همان بایت ماتریس حالت را تشخیص دهیم، این‌بار نیز یانیمهٔ بالای جدول معکوس *S-Box* انتخاب می‌شود و یا نیمهٔ پایین آن که این‌بار نیز ۱۲۸ عضو ممکن انتخاب می‌شوند. حتی اگر قسمت انتخاب شده از جدول معکوس *S-Box* تکراری باشد، از آنجا که دادهٔ ورودی متفاوت بوده، با *XOR* کردن فضای انتخابی با بایت متناظر از متن خوانا، یک فضای کلید ۱۲۸ عضوی جدید برای همان بایت از کلید به‌دست می‌آید که با فضای به‌دست آمدهٔ قبلی حدّاقل یک اشتراک دارد (کلید صحیح).

با تکرارروند بالا برای ورودی‌های مختلف، هر بار یک فضای کلید ۱۲۸ عضوی برای کلید به‌دست می‌آید که فضای کلید نهایی عبارت است از اشتراک همهٔ فضاهای کلید به‌دست آمده.

پس می‌توانیم روش حملهٔ خود را به‌صورت زیر مرتب کنیم:

۱. انتخاب یک متن خوانا
۲. اعمال متن به رمزنگار و نمونه‌برداری از توان مصرفی
۳. تشخیص بیت پرارزش یک بایت از ماتریس حالت با استفاده از شکل موج اثر توان در تبدیل مخلوط‌ساز ستونی دور اوّل
۴. استفاده از بیت تشخیص داده شده طبق (شکل ۱) برای یافتن فضای کلید
۵. اشتراک‌گیری بین فضای کلید به‌دست آمده و اشتراک فضاهای کلید قبلی

مراحل بالا را تکرار می‌کنیم تا فضای کلید به تعداد مورد نظر کاهش یابد. اگر تعداد تکرار کافی باشد، آنگاه فقط یک کلید به‌عنوان اشتراک همهٔ فضاهای کلید باقی می‌ماند که همان بایت کلید صحیح است.

۵- پیاده‌سازی عملی حمله

برای عملی‌کردن حملهٔ خود ابتدا الگوریتم *AES* را مطابق استاندارد *FIPS197* روی یک میکروکنترلر هشت‌بیتی *PIC* پیاده‌سازی کردیم و بدین ترتیب سخت‌افزار رمزکنندهٔ خود را آماده ساختیم. میکروکنترلر *PIC* محصول شرکت *Microchip* است وسهم به‌سزایی در بازار میکروکنترلرها و کارت‌های هوشمند دارد.

رمزکنندهٔ ساخته‌شده از طریق درگاه *USB* به رایانه متصل می‌شود. یک رابط کاربری[1] روی رایانه با استفاده از محیط *C#* از مجموعه *Visual Studio* تهیه کردیم که با رمزکننده ارتباط برقرار می‌کند و قالب‌های[2] ۱۲۸ بیتی داده را برای آن ارسال می‌کند. سخت‌افزار رمزکننده با دریافت یک قالب داده، آن را رمز می‌کند و نتیجه را به رایانه باز می‌گرداند. حال با داشتن سخت‌افزار به عملی‌سازی حملهٔ خود می‌پردازیم.

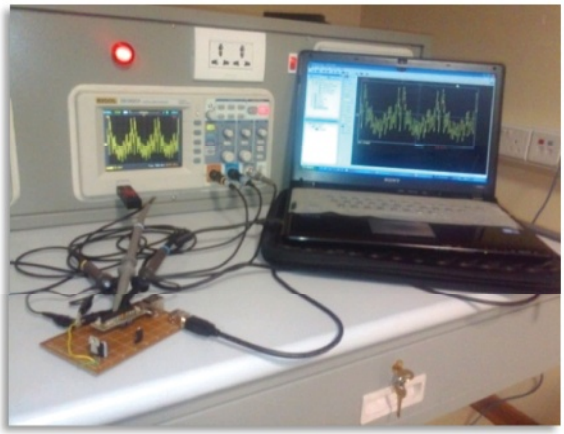
پیاده‌سازی حمله شامل دو مرحله است، یکی نمونه‌برداری و ذخیرهٔ اثرهای توان مصرفی رمزکننده حین رمزنگاری و سپس تشخیص بیت پر ارزش بایت‌های ماتریس حالت در تبدیل مخلوط‌ساز ستونی و دیگری استفاده از بیت تشخیص داده شده برای یافتن یک فضای کلید متناظر با دادهٔ ورودی.

برای نمونه‌برداری از اثر توان باید از یک اسلوسکوپ دیجیتال حافظه‌دار استفاده کنیم. در اینجا ما از اسیلوسکوپ *RIGOL DS1202CA* استفاده کرده‌ایم. این اسیلوسکوپ دارای پهنای باند *200MHz* و نرخ نمونه‌برداری *2GSa/S* است. همچنین دارای قابلیت اتصال به رایانه از طریق *USB* و *RS-232* است و همچنین می‌تواند سیگنال‌های گرفته شده را روی *USB Flash Memory* ذخیره کند. از دیگر امکانات مفید این اسیلوسکوپ قابلیت برنامه‌پذیری و کنترل از طریق رایانه است. کنترل این ابزار با استفاده از توابع موجود در *NI-VISA API* امکان‌پذیر است. این توابع زیرمجموعه‌ای از *VISA* هستند که به‌صورت استاندارد برای کار با وسایل اندازه‌گیری و آزمایشگاهی قابل اتصال به رایانه تدوین شده

[1] Application Interface
[2] Blocks

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]

است. رایانه دستورات را به‌صورت متوالی از طریق *USB* و یا *RS-232* به ابزار ارسال و ابزار تقاضای داده شده را اجرا می‌کند (*RIGOL , 2009*). از آنجا که در حملات تحلیل توان باید اثرهای توان زیادی را ذخیره کنیم، قابلیت برنامه‌پذیری اسیلوسکوپ برای سرعت بخشیدن به عملیات حمله بسیار مؤثّر خواهد بود. به‌منظور کنترل کردن اسیلوسکوپ از طریق رایانه نیز یک رابط کاربری با استفاده از *Visual C#* تهیه کردیم که با اسیلوسکوپ ارتباط برقرار کرده و در هنگام رمزنگاری، با ارسال دنبالهٔ دستورات مناسب به اسیلوسکوپ، اثر توان مصرفی را روی حافظه فلش ذخیره می‌کند. در (شکل ۲) تجهیزات حمله را مشاهده می‌کنید.



(شکل ۲): تجهیزات حمله شامل اسیلوسکوپ و رایانه

روش کلّی کنترل و هم‌زمان‌سازی اسیلوسکوپ و سخت‌افزار رمزکننده برای اعمال حمله این‌چنین است:

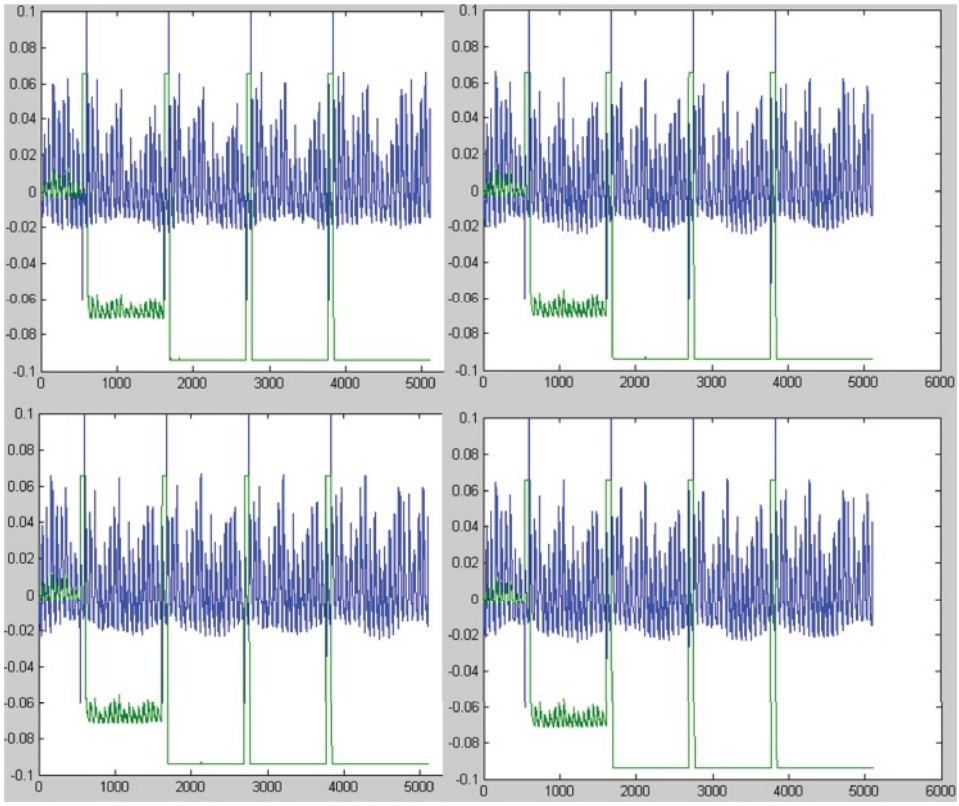
برای شروع یک نمونه‌برداری از سیگنال توان، ابتدا رایانه اسیلوسکوپ را (که حافظهٔ *USB Flash* به آن وصل است) آماده می‌کند و اسیلوسکوپ منتظر دریافت سیگنال چکانه[1] برای شروع نمونه‌برداری می‌ماند. این چکانه به‌صورت خارجی و از طریق رمزنگار تولید خواهد شد. علت این امر آن است که به محض شروع یک عملیات خاص، اسیلوسکوپ شروع به نمونه‌برداری کند و داده‌ای از دست نرود. حال رایانه دادهٔ مورد نظر برای رمزنگاری را به رمزنگار می‌فرستد. سپس به رمزنگار فرمان شروع می‌دهد. رمزنگار با دریافت این دستور عملیات خود را آغاز می‌کند و در ابتدای عملیات سیگنال، چکانه را نیز تولید می‌کند. با این کار اسیلوسکوپ شروع به نمونه‌برداری کرده و یک نمونه از اثر توان مصرف شامل ۵۱۲۰ نقطه برداشته می‌شود. وقتی رمزنگار عملیات خود را تمام کرد، اتمام کار خود را به رایانه اعلام می‌کند. در این لحظه رایانه با ارسال دنبالهٔ دستورات مناسب به اسیلوسکوپ، نمونهٔ توان برداشته‌شده را روی حافظهٔ فلش ذخیره و سپس اسیلوسکوپ را برای نمونه‌برداری بعدی آماده می‌کند. نمونه‌های توان لازم در این مقاله به‌ترتیب بالا و درحالی‌که فرکانس کاری سخت‌افزار رمزنگار *16MHz* بوده است تهیه شده‌اند. در حملات تحلیل توان برای داشتن یک نتیجهٔ خوب باید اثر نوفه را حذف کرد. برای حذف نوفه، عملیات رمزنگاری را برای یک دادهٔ مشخّص چندین‌بار تکرار کرده و اثرات توان را ذخیره می‌کنیم. سپس با انتقال داده‌ها به رایانه، از این اثرها میانگین‌گیری می‌کنیم. لازم به‌ذکر است که اسیلوسکوپ اطّلاعات خود را با قالب تک‌ستونی *CSV*[2] شامل ۵۱۲۰ نقطه از شکل موج نمونه‌برداری‌شده ذخیره می‌کند، میانگین‌گیری و رسم نمودارهای توان در محیط نرم‌افزار *MATLAB* انجام شده است. در نرم‌افزار *MATLAB* با استفاده از دستور *'importdata'* داده‌ها را به محیط نرم‌افزار فراخوانی می‌کنیم و پردازش‌های خود را روی آنها انجام می‌دهیم. در (شکل ۳) اثرات توان مصرفی سخت‌افزار رمزکننده حین اجرای تابع مخلوط‌ساز ستونی برای چهار بار تکرار عملیات رمزنگاری به‌ازای ورودی ثابت *'0x00'* آمده است. باتوجّه به شکل، می‌توان تفاوت‌هایی در اثرات ذخیره‌شده، به‌خصوص در قلّه‌ها، دید و این به‌دلیل تأثیر نوفه است.

بعد از تهیهٔ سیگنال‌ها و میانگین‌گیری و آماده‌سازی آنها باید به تشخیص بیت پرارزش بایت‌های ماتریس حالت در تابع مخلوط‌ساز ستونی دور اوّل بپردازیم. برای تشخیص بیت پرارزش باید اطّلاعاتی درخصوص پیاده‌سازی الگوریتم داشته باشیم. همان‌طور که درقبل گفته‌شد، اگر عمل *XOR* با *{1b}* بعد از عمل انتقال به چپ اجرا شود، می‌توان نتیجه گرفت که مقدار بیت پرارزش بایت داده قبل از انتقال به چپ *'1'* و در غیر این‌صورت *'0'* بوده است. پس اگر بتوانیم اجرا یا عدم اجرای *XOR* با *{1b}* را تشخیص دهیم، درواقع وضعیت بیت پرارزش بایت موردنظر را مشخّص کرده‌ایم. ممکن است که الگوریتم طوری پیاده‌سازی شده باشد که تعداد سیکل دستورالعمل در حالت *'1'* بودن بیت و اجرای *XOR* با تعداد سیکل دستورالعمل در حالت *'0'* بودن بیت و اجرا نشدن *XOR* یکسان نباشد. در این حالت به‌راحتی می‌توان با داشتن سیگنال توان مصرفی حین اجرای عمل ضرب بایت در *'02'* و مشاهده‌ی تعداد سیکل‌های دستورالعمل در اثر توان، حالتی را که بیت موردنظر *'1'* بوده و *XOR* اجرا شده را تشخیص داد. امّا در یک پیاده‌سازی امن‌تر می‌توان پیاده‌سازی را طوری انجام داد که در هر دو حالت صفر و یا یک بودن بیت، اجرای برنامه با تعداد سیکل ساعت یکسان

[1] Trigger

[2] Comma Separated Value

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]



(شکل ۳): اثرهای توان مصرفی حین اجرای تابع مخلوط‌ساز ستونی برای چهار بار تکرار عملیات رمزنگاری به‌ازای ورودی ثابت '0x00'

ادامه یابد. از آنجاکه ما از میکروکنترلر *PIC* به‌عنوان بستر پیاده‌سازی استفاده کرده‌ایم، پس باتوجّه به برگهٔ اطّلاعاتی این میکروکنترلر (*Microchip, 2007*) از دنبالهٔ دستورات اسمبلی زیر برای اجرای عمل ضرب در *'2'* به پیمانه *m(x)* استفاده می‌کنیم که شامل بررسی بیت پرارزش یک بایت و تصمیم‌گیری طبق آن برای اجرا یا عدم اجرای دستور بعد است:

```
RLNCF    WREG,f
BTFSC    WREG,0
XORLW    0x1a
```

دستور *RLNCF* ثبات *W* را یک بیت بدون جابه‌جاکردن نقلی به سمت چپ می‌چرخاند. حال دستور *BTFSC* بیت *0* از ثبات *W* را بررسی می‌کند. اگر این بیت، که قبل از چرخش بیت پرارزش *W* بود، صفر باشد، از دستور بعد پرش می‌شود که درواقع یک *NOP* به‌جای دستور بعد اجرا و بدین ترتیب در هر صورت (*'0'* و یا *'1'* بودن بیت مورد نظر از ثبات *W*) تعداد پالس‌های ساعت لازم مساوی خواهد شد. با وجود این پیاده‌سازی اگر دوباره سیگنال اثر توان مربوط به بررسی شرط یک‌بودن بیت پر ارزش ثبات *W* را ذخیره کنیم شکل‌هایی مانند (شکل۴) خواهیم داشت که اگر اثر نوفه را به‌خوبی حذف کرده باشیم، دوباره تحلیل‌هایی روی آنها می‌توانیم داشته باشیم.

در (شکل ۴ – الف) بیت مورد نظر از بایت اوّل، صفر است و به‌جای *XOR* یک *NOP* اجرا شده است. امّا در (شکل ۴ – ب) بیت مورد نظر یک بوده و *XOR* انجام شده است. در قسمت‌های مشخّص‌شده از دو سیگنال در (شکل۴)، تفاوت سطح توان مصرفی به‌خاطر این تفاوت تصمیم‌گیری مشخّص است.

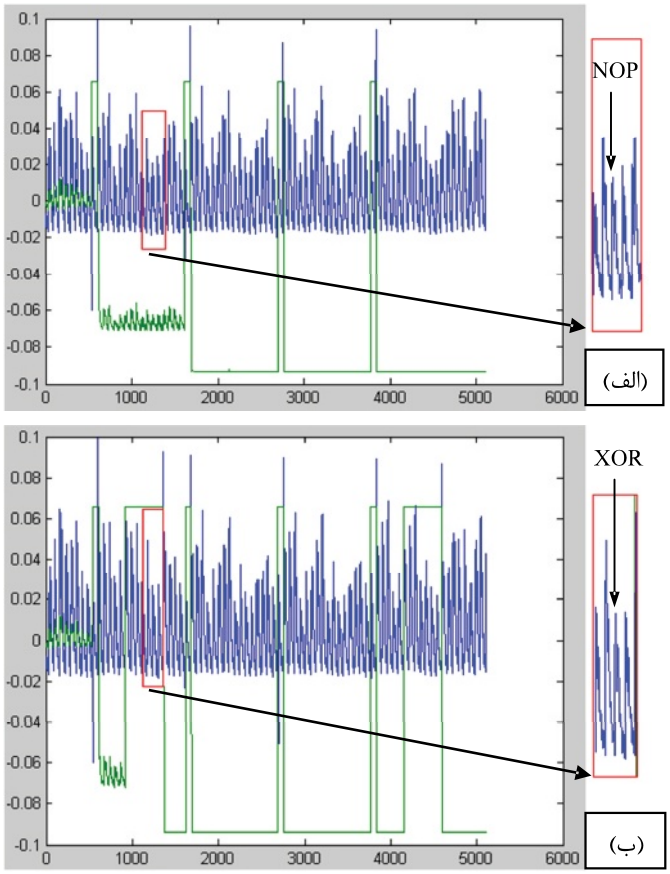
برای انجام این حمله، مراحل تحلیلی حمله و اشتراک‌گیری فضاهای کلید به‌دست آمده به‌صورت نرم‌افزاری انجام می‌شود. این نرم‌افزار در محیط *C++* تهیه شده است و با گرفتن دنباله بیت‌های به‌دست آمده به‌عنوان بیت پرارزش بایتی از ماتریس حالت در حین اجرای تابع مخلوط‌ساز ستونی دور اوّل در تکرارهای مجدد آزمایش با ورودی متفاوت، به محدودکردن فضای کلید می‌پردازد.

۶- نتایج عملی و تحلیل‌ها

در اینجا نتایج حاصل از اعمال حمله روی پیاده‌سازی و روند محدود شدن مرحله به مرحله‌ی فضای کلید را بیان می‌کنیم. این مراحل را برای بایت اول کلید عنوان می‌کنیم و برای دیگر بایت‌های کلید نیز مشابه همین روند وجود دارد.

در اوّلین آزمایش، بایت اوّل متن خوانا را *0x00* قرار داده و با شروع رمزنگاری حملهٔ خود را اعمال کردیم که

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]



(شکل۴): اختلاف در سطح توان عمل XOR و NOP

مقدار *'0'* برای بیت پرارزش از بایت اوّل ماتریس حالت در تبدیل مخلوط‌ساز ستونی حاصل شد. با دادن این مقدار به نرم‌افزار، ۱۲۸حالت زیر به‌عنوان فضای کلید ممکن معرفی شد:

key = 52, 9, 6a, d5, 30, 36, a5, 38, bf, 40, a3, 9e, 81, f3, d7, fb, 7c, e3, 39, 82, 9b, 2f, ff, 87, 34, 8e, 43, 44, c4, de, e9, cb, 54, 7b, 94, 32, a6, c2, 23, 3d, ee, 4c, 95, b, 42, fa, c3, 4e, 8, 2e, a1, 66, 28, d9, 24, b2, 76, 5b, a2, 49, 6d, 8b, d1, 25, 72, f8, f6, 64, 86, 68, 98, 16, d4, a4, 5c, cc, 5d, 65, b6, 92, c, 70, 48, 50, fd, ed, b9, da, 5e, 15, 46, 57, a7, 8d, 9d, 84, 90, d8, ab, 0, 8c, bc, d3, a, f7, e4, 58, 5, b8, b3, 45, 6, d0, 2c, 1e ,8f, ca, 3f, f, 2, c1, af, bd, 3, 1, 13, 8a, 6b.

در آزمایش دوم، بایت اوّل متن خوانا را *0x01* قرار داده و حملهٔ را تکرار کردیم که دوباره مقدار *'0'* برای بیت پرارزش از بایت اوّل ماتریس حالت در تبدیل مخلوط‌ساز ستونی به‌دست آمد. این‌بار نیز نرم‌افزار با گرفتن این مقدار، یک فضای ۱۲۸تایی برای کلیدهای ممکن به‌دست می‌آورد که اشتراک آن با فضای کلید آزمایش قبل، شامل ۶۴ عضو زیر است:

key = 9, 6a, d5, a5, 38, a3, fb, 39, 2f, 87, 8e, 43, 44, cb, 94, a6, c2, 95, b, 42, fa, c3, 8, 2e, d9, 24, b2, a2, 49, 6d, 8b, d1, 25, f6, 64, 86, d4, a4, 5c, 5d, 65, 6c, 48, b9, a7, 8d, d8, 0, 8c, bc, a, f7, b8, b3, 45, d0, 8f, ca, 2, bd, 3, 1, 8a, 6b.

در تلاش سوم بایت اوّل متن خوانا را *0x02* انتخاب می‌کنیم. این‌بار نیز *'0'* برای مقدار بیت پرارزش به‌دست آمد.

در این مرحله اشتراک فضاهای کلید شامل سی عضو است:

key = 9, 6a, d5, a5, a3, 8e, 44, a6, b, 42, fa, c3, 8, 2e, d1, 64, 86, a4, 5c, a7, 8d, d8, 0, 8c, a, 8f, 2, bd, 3, 1.

به همین ترتیب تلاش‌های خود را ادامه دادیم تا این‌که در آزمایش سیزدهم تنها به یک کلید رسیدیم. در زیر نتایج این مراحل به‌ترتیب آمده است:

tray : 4, text[0]=0x03, bit=0, n=16
key = 9, a5, 8e, a6, b, 8, a4, a7, 8d, 0, 8c, a, 8f, 2, 3, 1.
tray : 5, text[0]= 0x04, bit=1, n=8
key = 9, 8, a4, 8d, 0, 8c, a, 3.
tray : 6, text[0]= 0x05, bit=0, n=4
key = a4, 0, a, 3.
tray : 7, text[0]= 0x06, bit=0, n=3
key = a4, 0, 3.
tray : 8, text[0]= 0x07, bit=1, n=2
key = 0, 3.

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]

tray : 9 , text[0]= 0x08, bit=0, n=2
key=0, 3.
tray : 10, text[0]= 0x09, bit=0, n=2
key=0, 3.
tray : 11, text[0]= 0x0a, bit=0, n=2
key = 0, 3.
tray : 12, text[0]= 0x0b, bit=0, n=2
key = 0, 3.
tray : 13, text[0]= 0x0c, bit=1, n=1
key = 0.

همان‌طور که مشاهده می‌شود، درنهایت تنها عضو مشترک همهٔ فضاهای به‌دست آمده، عبارت است از *'0x00'* که مقدار صحیح برای بایت اوّل کلید استفاده شده در سخت‌افزار در زمان انجام این حمله بود.

این حمله برای مقادیر مختلفی از بایت‌های کلید آزمایش شده و همواره کلید را به‌درستی نتیجه گرفته است. طبق نتایج حاصل از پیاده‌سازی این حمله به این نتیجه می‌رسیم که با داشتن حدّاکثر شانزده متن خوانا، در هر صورت اشتراک فضاهای کلید تنها شامل یک عضو خواهد شد که همان کلیدرمزنگاری است.

۷- راه‌کار پیشنهادی برای مقابله با این نقطه‌ضعف

به‌طور معمول ضعف سامانه‌ها در برابر *SPA*، به‌دلیل وجود عمل‌گر شرطی در ساختار آنها و بی‌دقتی در پیاده‌سازی این عملگرهای شرطی است. در این مقاله نیز ما از وجود اینچنین ضعفی در پیاده‌سازی تابع مخلوط‌ساز ستونی استفاده کردیم. درواقع وجود عملگر شرطی در ساختار برنامه می‌تواند باعث ایجاد تفاوت قابل کشف در شکل موج توان توسط *SPA* شود. به‌عنوان پیشنهادی برای مقابله با این حمله، می‌توان ساختار شرطی برای محاسبهٔ *'XTime'* که درواقع همان عمل ضرب یک بایت در *'2'* در $GF(2^8)$ است را کنار گذاشت و به‌جای آن از جدول جستجو[1] برای یافتن این حاصل‌ضرب استفاده کنیم. بدین ترتیب در هنگام پیاده‌سازی مخلوط‌ساز ستونی، در مواقعی که لازم است حاصل‌ضرب یک بایت در *'2'* را بدانیم، کافی است باتوجّه به بایت مورد نظر، مقدار متناظر با آن‌را از جدول جستجو فراخوانی کرده و ادامهٔ برنامه را پیگیری کنیم. از آنجاکه هر بایت شامل ۲۵۶ حالت است، جدول جستجو نیز شامل ۲۵۶ حاصل‌ضرب ممکن برای ضرب یک بایت داده در *'2'* به‌صورت ضرب چندجمله‌ای‌های باینری به پیمانهٔ چندجمله‌ای تحویل‌ناپذیر $X^8+X^4+X^3+X+1$ روی $GF(2)$ خواهد بود.

[1] Look-up Table

پس تنها تغییر برنامه برای مقاوم‌سازی در برابر این حملهٔ *SPA*، تغییردادن تابع *'XTime'* به‌گونه‌ای است که حاصل‌ضرب از جدول جستجو فراخوانی شود.

۸- جمع‌بندی و نتیجه‌گیری

در این مقاله ضمن ارائه مقدمه‌ای از حمله‌های کانال جانبی به‌خصوص حمله‌های تحلیل توان و مرور ساختار الگوریتم *AES* به ضعف موجود در تابع مخلوط‌ساز ستونی این الگوریتم در برابر *SPA* پرداختیم و نشان دادیم که چگونه می‌توان از این ضعف برای اعمال یک حملهٔ تحلیل توان ساده به *AES* استفاده کرد. سپس روش عملی‌کردن این حمله را روی یک پیاده‌سازی *AES* بر روی میکروکنترلر *PIC* آوردیم. همچنین تجهیزات سخت‌افزاری و نرم‌افزاری لازم به همراه نتایج عملی و نحوهٔ بازیابی کلید بیان شد. حمله‌های تحلیل توان همواره یکی از تهدیدهای مهم برای سامانه‌های رمزنگاری هستند. برای مقابله با حمله *SPA* در پیاده‌سازی‌ها به این نکته باید دقّت کرد که از وجود ساختارهای شرطی در الگوریتم اجتناب شود.

۹- منابع

Chen, C.N., Yen, S.M., 2003. "Differential Fault Analysis on AES Key Schedule and Some countermeasures," ACISP'03 Proceedings of the 8th Australasian conference on Information security and privacy, LNCS2727, pp.118-129, Wollongong, Australia, Jul. 2003, Springer-Verlag.

Daemen, J., Rijmen, V., 2000. Rijndael Block Cipher, AES Proposal, Computer Security Resource Center, National Institute of Standards and Technology.

Gandolfi, K., Naccache, D., Paar , C., Karine, G., Mourtel,C., Olivier, F., 2001. "Electromagnetic Analysis: Concrete Results," CHES 2001, Third Intemotional Workshop, Paris, France, May 14-16,2001, Proceedings, LNCS2162, pages 251-261. Springer.

Goldack, M.,2008.Side-Channel based Reverse Engineering for Microcontrollers, Ruhr-University.

Koeune, F., Quisquater, J.J., 1999. "A Timing Attack against Rijndael," Technical Report CG-1999/1 ,UniversiteCatholique de Louvain, Available online at ht tp:// www.di ce.ucl .ac.b e/ cry pt o / techreports .html

Kocher, P., 1996. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," CRYPTO'96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings, LNCS1109, pages 104-113. Springer-Verlag London.

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]

Kocher, P., Jaffe, J., Jun, B., 1999. "Differential Power Analysis,"CRYPTO'99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, LNCS1666, pages 388-397, Springer.

Mangard, S., 2003.A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion, Springer-Verlag Berlin Heidelberg.

Mangard, S., 2004. Securing Implementations of Block Ciphers against Side-Channel Attacks, Ph.D. Thesis, Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Austria.

Mangard, S.,Oswald, E.,Popp, T.,2007. Power Analysis Attacks-Revealing the Secrets of Smart Cards, Springer Science+Business Media.

Masoomi, M., Masoumi, M., Ahmadian, M., 2010 "A Practical Differential Power Analysis Attack against an FPGA Implementation of AES Cryptosystem," IEEE International Conference on Information Society (I-Society 2010), United Kingdom, London, 28-30 June.

Microchip Technology Inc, 2007. PI C18F 245 5/25 50/4455/4550 Data Sheet.

NIST. 2001. ADVANCED ENCRYPTION STANDARD (AES), Federal Information Processing Standards Publication 197, November 26.

Quisquater, J. J., Samyde, D., 2001. "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards," International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings, LNCS2140 , pages 200-210. Springer.

RIGOL Technologies Inc, 2009. RIGOL Programming Guide DS1000CA Series Digital Oscilloscope.

Samyde, D.,Skorobogatov, S.,Anderson, R.,Quisquater, J.,2002. On a New Way to Read Data from Memory. In IEEE Security in Storage Workshop (SISW'02), IEEE Computer Society, pp. 65-69.

Schmidt, J., Kim, C.H., 2009. "A Probing Attack on AES", WISA 2008, LNCS 5379, Springer-Verlag Berlin Heidelberg, pp. 256-265.

Skorobogatov, S., Anderson, R., 2003. "Optical Fault Induction Attacks," CHES '02 Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems, LNCS2523, pages 2-12. Springer.

Skorobogatov. S., 2005. Semi-invasive attacks - A new approach to hardware security analysis. PhD thesis, University of Cambridge, Available online at http://www.cl.cam.ac.uk/TechReports/.

Whelan, C., Tunstall, M., Choukri, H., Naccache, D., Bar-El, H., 2006. "The Sorcerer's Apprentice Guide to Fault Attacks," Proceedings of the IEEE, Special Issue on Cryptography and Security, Volume 96, Number 2, pages 370-382.



حامد یوسفی در سال ۱۳۸۸ مدرک کارشناسی خود را در رشتهٔ مهندسی برق گرایش الکترونیک از دانشگاه شهرکرد دریافت کرد و مدرک کارشناسی ارشد خود را در رشتهٔ مهندسی مخابرات گرایش رمز از دانشگاه جامع امام حسین(ع) در سال ۱۳۹۰ دریافت نمود. زمینه‌های مورد علاقهٔ وی امنیت سخت‌افزاری و سامانه‌های تعبیه شده است.

نشانی رایانامهٔ ایشان عبارتست از:

h.yusefi@rcisp.com



محمود گردشی کارشناسی خود را از دانشگاه شیراز، کارشناسی ارشد را از دانشگاه تبریز و دانشوری را از دانشگاه امیرکبیر به ترتیب در سال‌های ۱۳۶۸، ۱۳۷۰ و ۱۳۷۸ اخذ نموده‌اند و درحال حاضر عضو هیئت علمی دانشگاه جامع امام حسین(ع) هستند. زمینهٔ مورد علاقهٔ وی پروتکل‌های رمزنگاری و سامانه‌های کلید عمومی است.

نشانی رایانامهٔ ایشان عبارتست از:

mgardeshi2000@yahoo.com



محمد سبزی‌نژاد فراش درجهٔ کارشناسی خود را در رشتهٔ مهندسی برق گرایش الکترونیک از دانشکدهٔ فنی شهید چمران کرمان در سال ۱۳۸۵ دریافت نمود. او درجهٔ کارشناسی ارشد را در رشتهٔ مهندسی مخابرات رمز از دانشگاه جامع امام حسین(ع) در سال ۱۳۸۷ اخذ نمود. او از سال ۱۳۸۸ تا کنون دانشجوی دکتری ریاضیات رمز دانشکدهٔ علوم ریاضی و کامپیوتر دانشگاه تربیت معلم می‌باشد. پروتکل‌های رمزنگاری، مدل‌های اثبات امنیتی، امنیت شبکه و امنیت سخت افزاری از جمله علاقمندی‌های تحقیقاتی وی است.

نشانی رایانامهٔ ایشان عبارتست از:

sabzinejad@tmu.ac.ir

[Downloaded from jsdp.rcisp.ac.ir on 2026-11-04]