بررسی حملات کانال جانبی حافظه نهان بر روی پیاده‌سازی الگوریتم رمز AES

محمد جدیدی[۱]، مهدی اصفهانی[۲]، اسماعیل محمدقلی[۳]

۱- مربی- دانشکده کامپیوتر و قدرت سایبری - دانشگاه جامع امام حسین ع - تهران- ایران
mohammad.jadidi@gmail.com

۲- دکتری ریاضی کاربردی و پژوهشگر آزمایشگاه ISSL دانشکده برق- دانشگاه صنعتی شریف - تهران- ایران
maahdisf2000@yahoo.com

۳- پژوهشگر دانشکده کامپیوتر و قدرت سایبری - دانشگاه جامع امام حسین ع - تهران- ایران
esmail.mohammadgholi@gmail.com

چکیده: با پیشرفت صنعت الکترونیک و پیدایش پردازنده‌های مدرن، مدل حمله در الگوریتم‌ها و پروتکل‌های رمزنگاری نیز تغییر کرد. با وجود پیچیدگی محاسباتی در الگوریتم‌ها و پروتکل‌های رمزنگاری، پیاده‌سازی‌ها می‌توانند عاملی برای نشت اطلاعات محرمانه باشند. مهاجم می‌تواند زمانی حمله کند که قطعات الکترونیکی در حال اجرای عملگرهای رمزنگاری با استفاده از کلید مخفی بر روی داده‌های حساس هستند. در حین رایانش، نشت اطلاعات در قطعات الکترونیکی وجود دارد که به آن حملات کانال جانبی می‌گویند. یکی از مهم‌ترین منابع نشت اطلاعات کانال جانبی، تغییرات زمانی ناشی از اجرای محاسبات است. دسترسی‌ها به حافظه و وجود انشعاب‌ها در برنامه، در زمان اجرا هزینه‌بر هستند، بنابراین پردازنده‌ها برای کاهش این هزینه، از حافظه نهان و پیشگویی انشعاب استفاده می‌کنند. متأسفانه این بهینه‌سازی در زمان اجرا، منجر به ایجاد تغییرات زمانی در اجرای یک برنامه می‌شود. حافظه نهان در حملات کانال جانبی زمان، چالش‌برانگیزتر و کاربردی‌تر است. در این مقاله به مرور انواع حملات حافظه نهان روی پیاده‌سازی الگوریتم رمز AES خواهیم پرداخت. با پیاده‌سازی حملات و مقایسه نتایج، ضعف‌های امنیتی پیاده‌سازی الگوریتم رمز AES در برابر حملات حافظه نهان را استخراج و مورد مقایسه قرار خواهیم داد.

واژه‌ای کلیدی: حملات کانال جانبی حافظه نهان – امنیت اطلاعات – حملات زمان‌محور – حملات مبتنی بر دسترسی به حافظه

تاریخ ارسال مقاله : ۱۳۹۹/۱۰/۰۲
تاریخ پذیرش مقاله : ۱۴۰۰/۰۱/۱۵
نام نویسنده مسئول: محمد جدیدی

۱- مقدمه

یکی از تهدیدات موجود که می‌تواند منجر به نشت اطلاعات شود، هنگام اجرای عملگرهای رمزنگاری با استفاده از کلید مخفی بر روی داده‌های حساس در قطعات الکترونیکی است. به حملاتی که از نشت اطلاعات مخفی در قطعات الکترونیکی حین اجرا بهره می‌برد، حملات کانال جانبی گویند. پیاده‌سازی ناامن یک طرح رمزنگاری می‌تواند موجب آسیب‌پذیری طرح در برابر حملات کانال جانبی شود. اخیراً یکی از حملات کانال جانبی که به دلیل خطرناک بودن آن بسیار مطرح است و کمتر روی آن کارشده، حملات کانال جانبی زمانی [۱] مبتنی بر معماری پردازنده‌ها است. در این حملات، برخی از اجزا پردازنده به‌دلیل پیاده‌سازی نرم‌افزاری ناامن یک طرح رمزنگاری، می‌توانند موجب نشت اطلاعات زمانی شوند. بنابراین شناخت کامل معماری پردازنده‌ها مانند اینتل و ARM امری لازم است. نشت اطلاعات زمانی در سیستم‌های رمزنگاریِ فضای ابر [۲]، ماشین‌های مجازی (VMs) [۳]، سیستم‌های چندکاربره مانند سیستم‌عامل اندروید در موبایل‌های هوشمند [۴] و کتابخانه‌ی رمزنگاری OpenSSL [۵]، نشانگر تهدید جدی و عملی حملات زمانی علیه سامانه‌های امنیتی نسبت به سایر حملات کانال جانبی است.

حملات کانال جانبی زمانی به حملات راه‌دور و راه نزدیک دسته‌بندی می‌شوند. حملات زمانی راه دور بیشتر برای رمزهای کلید عمومی و پروتکل‌ها قابل انجام است [۵]. حملات راه نزدیک، مبتنی بر اجزایی از پردازنده است که هنگام اجرای رمز، نشت اطلاعات زمانی در آنها اتفاق می‌افتد و به حملات مبتنی بر پیشگویی انشعاب و حافظه نهان دسته‌بندی می‌شوند. واحد پیشگویی انشعاب یکی از اجزای پردازنده‌های مدرن است (به‌منظور کاهش سربار عملکرد برنامه وقتی یک انشعاب اتفاق می‌افتد) می‌تواند به‌طور قابل‌ملاحظه‌ای در زمان اجرای رمز نفوذ کرده و مهاجم از این موضوع می‌تواند برای حمله به رمزهای کلید عمومی استفاده نماید. حافظه‌ی نهان نیز یکی دیگر از اجزای پردازنده‌های مدرن است که می‌تواند موجب نشت اطلاعات زمانی، بالاخص برای رمزهای قالبی باشد. در حملات حافظه نهان، می‌بایست دسترسی‌ها به حافظه هنگام اجرای رمز در مکان‌هایی رخ دهد که وابسته به کلید مخفی باشند [۶].

در ابتدای این پژوهش دسته‌بندی حملات کانال جانبی زمانی موردبررسی قرار می‌گیرد. با توجه به ساختار حافظه‌ی نهان، حملات کانال جانبی زمانی مبتنی بر حافظه نهان کاربردی‌تر بوده و می‌تواند تهدید جدی و عملی علیه سامانه‌های امنیتی نسبت به سایر حملات کانال جانبی زمانی داشته باشد. بر اساس میزان دسترسی مهاجم، حملات کانال جانبی حافظه نهان را می‌توان به دو دسته حملات زمان‌محور و حملات مبتنی بر دسترسی به حافظه دسته‌بندی نمود. در حملات زمان‌محور، مهاجم صرفاً توانایی رصد کل زمان اجرای الگوریتم رمزنگاری را دارد [۷, ۸]. در مقابل در حملات مبتنی بر دسترسی به حافظه، فرض می‌شود که مهاجم توانایی ایجاد تغییرات در حافظه نهان را نیز دارد [۹-۱۱]. حملات زمان‌محور شامل روش‌های مبتنی بر تصادم‌های داخلی [۷]، مبتنی بر خواص الگوریتم‌های رمز [۱۲] و روش نمایه شده [۱۳] بوده و حملات مبتنی بر دسترسی به حافظه با استفاده از روش‌های Prime+Prob [۹]، Evict+Time[۱۴]، Flush+Reload[۱۵, ۱۶] و Flush+Flush[۱۷] انجام می‌شوند.

در ادامه مقاله مبانی و مفاهیم اولیه در خصوص پیاده‌سازی نرم‌افزاری الگوریتم AESو ساختار حافظه نهان مورد بررسی قرار می‌گیرد. در بخش ۳ مروری بر حملات کانال جانبی زمان و در بخش ۴ حملات حافظه نهان و انواع آن مورد بررسی قرار می‌گیرد. در بخش ۵ نتایج عملی پیاده‌سازی حملات حافظه نهان بر روی الگوریتم AES مورد مقایسه و بررسی قرار می‌گیرد.

۲- مبانی و مفاهیم اولیه

در این بخش پیاده‌سازی نرم‌افزاری AES و ساختار حافظه نهان مورد بررسی قرار می‌گیرد.

۲-۱- پیاده‌سازی نرم‌افزاری AES مبتنی بر جداول مبنا

رمز AES نمونه‌ای از ساختار شبکه جانشینی- جایگشتی در رمزهای قالبی است که در سال ۲۰۰۰ تحت عنوان رایندل در مسابقه‌ی AES انتخاب و در سال ۲۰۰۱ تحت عنوان رمز AES، توسط موسسه ملی استاندارد و فنّاوری آمریکا به‌عنوان یک الگوریتم استاندارد انتخاب شد. رمز AES از کلیدهایی به طول ۱۲۸، ۱۹۲ و ۲۵۶ بیتی برای رمزگذاری و رمزگشایی استفاده می‌کند که در این مقاله همانند مقالات پیشین تنها حمله به الگوریتم رمزنگاری AES با طول کلید ۱۲۸ را موردبررسی قرار می‌دهیم.

در AES داده‌ها به‌صورت بایتی نمایش داده می‌شوند که هر بایت درواقع عضوی از میدان $GF(2^8)$ ت. . ودی رمز AES-128، به‌صورت یک ماتریس ۴×۴ بایتی مرتب می‌شود که به آن ماتریس حالت گویند. AES با طول کلید ۱۲۸ از ده دور تشکیل شده است. تابع دور از چهار تبدیل جانشینی بایتی، فت سطری، مخلوط‌سازی ستونی ی تبدیل جمع با زیرکلید دور ر کیل شده است.

برای افزایش سرعت و کارآیی در اجرای الگوریتم‌های رمزنگاری، روش‌های پیاده‌سازی در سخت‌افزار و نرم‌افزار تفاوت دارند. به‌عنوان یک روش کارا و پرکاربرد، در پیاده‌سازی نرم‌افزاری رمزهای قالبی، ترکیب جعبه‌های جانشینی و لایه‌ی خطی با استفاده از جداول مبنا پیاده‌سازی می‌شوند. از مزایای استفاده از این جداول در پیاده‌سازی نرم‌افزاری، می‌توان به انعطاف‌پذیر بودن بخش‌هایی از پیاده‌سازی و سرعت بالای اجرا اشاره کرد. در رمز AES، چهار جدول مبنا برای ترکیب لایه‌ی جانشینی بایتی و لایه خطی مخلوط‌ساز ستونی به‌صورت رابطه‌ی (۱) ساخته می‌شوند [۱۸] که در مقالات و نوشته‌های علمی

عموماً تحت عنوان T-table نام‌گذاری می‌شوند و در کتابخانه‌های معروف رمزنگاری مانند OpenSSL مورداستفاده قرار می‌گیرند.

$$T_0[z]=\begin{bmatrix}02.S(z)\\S(z)\\S(z)\\03.S(z)\end{bmatrix}\quad T_1[z]=\begin{bmatrix}03.S(z)\\02.S(z)\\S(z)\\S(z)\end{bmatrix}$$
$$T_2[z]=\begin{bmatrix}S(z)\\03.S(z)\\02.S(z)\\S(z)\end{bmatrix}\quad T_3[z]=\begin{bmatrix}S(z)\\S(z)\\03.S(z)\\03.S(z)\end{bmatrix}$$

(۱)

با توجه به جداول ارائه‌شده در رابطه‌ی (۱)، نه دور اول AES می‌تواند به‌صورت رابطه‌ی (۲) محاسبه شود.

$$\begin{aligned}&T_0[s_0^{(r)}]\oplus T_1[s_5^{(r)}]\oplus T_2[s_{10}^{(r)}]\oplus T_3[s_{15}^{(r)}]\oplus[k_0^{(r)}k_1^{(r)}k_2^{(r)}k_3^{(r)}]\\&T_0[s_4^{(r)}]\oplus T_1[s_9^{(r)}]\oplus T_2[s_{14}^{(r)}]\oplus T_3[s_3^{(r)}]\oplus[k_4^{(r)}k_5^{(r)}k_6^{(r)}k_7^{(r)}]\\&T_0[s_8^{(r)}]\oplus T_1[s_{13}^{(r)}]\oplus T_2[s_2^{(r)}]\oplus T_3[s_7^{(r)}]\oplus[k_8^{(r)}k_9^{(r)}k_{10}^{(r)}k_{11}^{(r)}]\\&T_0[s_{12}^{(r)}]\oplus T_1[s_1^{(r)}]\oplus T_2[s_6^{(r)}]\oplus T_3[s_{11}^{(r)}]\oplus[k_{12}^{(r)}k_{13}^{(r)}k_{14}^{(r)}k_{15}^{(r)}]\end{aligned}$$

(۲)

در رابطه (۲)، ماتریس حالت در دور rام را با $s^{(r)}$، iامین بایت از ماتریس حالت در دور rام را با نماد $s_i^{(r)}$ نشان می‌دهیم که در آن $0\le r\le 9$ و $0\le i\le 15$ می‌باشند.

به دلیل نبود عملگر مخلوط‌ساز ستونی در دور آخر (دور دهم الگوریتم)، نمی‌توان به‌صورت ذکرشده در رابطه (۲) از این جداول استفاده کرد. در پیاده‌سازی‌های استاندارد عموماً دو راهکار برای پیاده‌سازی دور آخر استفاده می‌شود. یک راه استفاده از یک جدول مبنای مانند T_4 است. راهکار دیگر استفاده از جداول ذکرشده در رابطه (۱) به نحوی متفاوت است. از آنجائی که دور آخر تنها شامل جعبه جانشانی و شیفت سطری است، می‌توان مقادیر $S(s_{12}^9)$، $S(s_8^9)$، $S(s_4^9)$ و $S(s_0^9)$ را با فراخوانی T_0 و استفاده از (تنها) بایت دوم آن محاسبه کرد. به‌طور مشابه می‌توان سایر بایت‌ها را با فراخوانی جداول T_1، T_2 و T_3 محاسبه کرد. لازم به ذکر است که پیاده‌سازی مورداستفاده در کتابخانه OpenSSl نسخه‌ی 1.1.0 f از روش دوم به‌منظور پیاده‌سازی دور آخر استفاده کرده است. به‌عبارت‌دیگر، هر یک از جداول T، در هر دور ۴ بار و درمجموع به ازای یک عمل رمزنگاری ۴۰ بار فراخوانی می‌شوند.

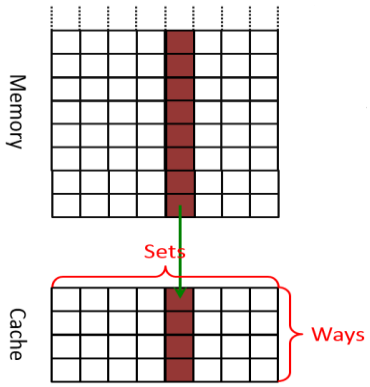
۲-۲- حافظه نهان

در پردازنده‌های مدرن، حافظه نهان سرعت دسترسی به حافظه را افزایش می‌دهد که این کار را به‌وسیله‌ی نگهداری داده‌ها و دستورالعمل‌هایی که اخیراً مورد دسترسی قرار گرفتند، انجام می‌دهد. حافظه نهان، یک بانک از حافظه با سرعت بالا بوده که نسبت به حافظه‌ی اصلی رایانه سریع‌تر و گران‌تر است [۱۹].

هر بخش از حافظه‌ی نهان از بخش‌هایی به نام خطوط کش تشکیل شده است. هر خط از سه عنصر بلوک داده، تگ و بیت اعتبارسنجی تشکیل شده است. بلوک داده شامل اطلاعاتی است که هنگام دسترسی پردازنده به آن، از حافظه‌ی اصلی به حافظه نهان کپی می‌شود. تگ، آدرسی از حافظه‌ی اصلی است که داده در آن قرار دارد و بیت اعتبارسنجی یک فیلد تک‌بیتی است که اعتبارِ داده‌ی قرارگرفته در خط کش را می‌سنجد.

در حالتی که فقدان کش رخ دهد پردازنده، داده یا دستورالعمل را از حافظه‌ی اصلی می‌خواند و آن را در حافظه نهان بارگذاری می‌کند. سه روش رایج برای نگاشت داده از حافظه‌ی اصلی به حافظه نهان وجود دارد. در روش اول، حافظه‌ی اصلی به بلوک‌های مساوی تقسیم می‌شود و هرکدام از خانه‌های حافظه نهان فقط به یکی از بلوک‌های حافظه تخصیص می‌یابد که به آن نگاشت مستقیم حافظه نهان می‌گویند. مینیمم زمان جستجو از مزایا و بدترین عملکرد به دلیل استفاده نشدن احتمالی قسمت‌هایی از حافظه نهان، از معایب این روش است. در روش دوم، مقدار یک خانه از حافظه‌ی اصلی، در هر خانه‌ی دلخواه از حافظه نهان می‌تواند ذخیره شود که به آن نگاشت شرکت‌پذیر کامل حافظه نهان می‌گویند. بهترین عملکرد از مزایا و بدترین حالت در زمان جستجو از معایب این روش است. در روش سوم، هر خانه از حافظه‌ی اصلی فقط می‌تواند در یک تعداد مشخصی از خانه‌های حافظه نهان ذخیره شوند که به آن مجموعه‌ی شرکت‌پذیر حافظه نهان می‌گویند. بهترین حالت برای زمان جستجو و عملکرد حافظه نهان نسبت به دو روش قبلی، موجب شده ریزپردازنده‌های مدرن از روش سوم استفاده کنند [۲۰].

حافظه نهان به یک سری مجموعه‌هایی تقسیم‌بندی می‌شود و هر مجموعه شامل یک تعدادی سوی است که یک بلوک از حافظه می‌تواند در هر سوی از یک مجموعه ذخیره شود (شکل ۱).

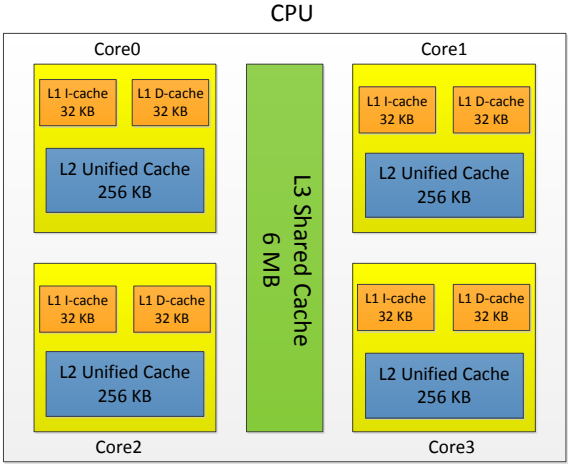


شکل(۱): ساختار حافظه نهان

در پردازنده‌هایی همچون پردازنده‌های اینتل، سلسله‌مراتب در حافظه نهان شامل سه سطح است. بالاترین سطح را L1 می‌نامند که نزدیک‌ترین سطح به هسته بوده و کم‌حجم‌ترین و بالاترین سرعت را دارد. پایین‌ترین سطح حافظه نهان LLC است که کندترین سرعت و بیشترین حجم را دارا می‌باشد [۲۱].

لایه L1 به دو قسمت تقسیم می‌شود: L1-D که داده‌های برنامه در آن ذخیره می‌شود و L1-I که دستورالعمل‌ها و کدهای اجرای برنامه در آن ذخیره می‌شود. در پردازنده‌های چندهسته‌ای، معمولاً هر هسته

شامل لایه‌های مجزا L1 و L2 بوده و همچنین لایه LLC بین همه‌ی هسته‌ها به اشتراک گذاشته می‌شود (شکل ۲) [۲۳]. در بیشتر پردازنده‌های اینتل رابطه بین لایه‌های حافظه نهان بدین‌صورت است که تمامی مقادیر ذخیره‌شده در L1 در لایه‌های L2 و LLC نیز وجود دارند. همچنین تمامی مقادیر ذخیره‌شده در L2 در لایه LLC نیز وجود دارند. به این خاصیت اصطلاحاً inclusive می‌گویند.



شکل (۲): پردازنده‌ی چهار هسته‌ای

۳- حملات کانال جانبی زمان

پیاده‌سازی ناامن یک طرح رمزنگاری می‌تواند موجب آسیب‌پذیر بودن طرح در برابر حملات کانال جانبی شود. اخیراً یکی از حملات کانال جانبی که به دلیل خطرناک بودن آن بسیار مطرح است و کمتر روی آن کار شده، حملات کانال جانبی زمانی است. حملات زمانی از این واقعیت استفاده می‌کند که زمان لازم برای اجرای یک عملیات، وابسته به ورودی رمز بوده که این ورودی‌ها شامل متن آشکار و کلید است. همچنین انجام یک سری از عملیات ریاضی قبل از رسیدن به پیام رمزشده وجود دارد، این عملیات ریاضی به همراه مواردی مانند عبارات شرطی، پیشگویی انشعاب، تعداد و نوع دستورات اجراشده، عملگرها و مکان‌های حافظه‌ی مورد دسترسی توسط برنامه در طول اجرای رمز، در زمان اجرا تأثیرگذار هستند. این عوامل در سیستم می‌توانند به‌طور بالقوه، اطلاعاتی را درباره‌ی کلید و متن آشکار به‌وسیله‌ی حمله‌ی کانال جانبی زمانی بازیابی کنند.

برخی از اجزای پردازنده که به آنها ریزمعماری هم گفته می‌شود، هنگام درگیر شدن در اجرای رمز، می‌توانند عاملی جهت نشت اطلاعات زمانی باشند. حافظه‌ی نهان یکی از اجزای پردازنده‌های مدرن است که می‌تواند موجب نشت اطلاعات زمانی، بالأخص برای رمزهای قالبی باشد. در حملات حافظه نهان، می‌بایست دسترسی‌ها به حافظه هنگام اجرای رمز در مکان‌هایی رخ دهد که وابسته به کلید مخفی باشند. همچنین این حملات از این حقیقت استفاده می‌کنند که فقدان حافظه نهان در مقایسه با برخورد حافظه نهان، زمان بیشتری را صرف می‌کند [۷].

۴- حملات کانال جانبی زمان مبتنی بر حافظه نهان بر روی پیاده‌سازی الگوریتم رمز AES

کانال‌های جانبی مبتنی بر زمان در مقایسه با کانال‌های جانبی دیگر کم‌تر شناخته شده‌اند و ویژگی‌هایی دارند، مانند: اندازه‌گیری آسان و عدم نیاز به ابزار سخت‌افزاری خاص برای اندازه‌گیری. در یک کانال جانبی زمانی، رویدادهای ریزپردازنده مانند برخورد حافظه نهان و فقدان حافظه نهان با استفاده از اندازه‌گیری‌های زمان متمایز می‌شوند. برای اندازه‌گیری زمان دسترسی به منابع دقیق کلاک در پردازنده الزامی است. خیلی از پردازنده‌ها، کلاک‌های دقیقی به فرم شمارنده‌ی مهر زمانی[1] دارند، که دقت خیلی بالا و سربار کم برای اندازه‌گیری زمان هستند. به‌طور مثال در ماشین‌های اینتل مانند پنتیوم، یک ثبات ۶۴ بیتی برای شمارنده‌ی مهر زمانی وجود دارد. وقتی پردازنده ریست شود ثبات مقدار صفر را دارد و بعد به‌طور یکنواخت در هر کلاک سیکل مقدار آن افزایش می‌یابد. بنابراین، روی یک ماشین 1 GHz، شمارنده‌ها می‌توانند زمان را در حد نانوثانیه اندازه‌گیری کنند. در زبان اسمبلی یک دستوری به نام rdtsc وجود دارد که شمارنده‌ی مهر زمانی را می‌خواند. وقتی rdtsc فراخوانی می‌شود بالاترین مرتبه‌ی بیت‌های شمارنده (۳۲ بیتی) در ثبات edx بارگذاری شده و پایین‌ترین مرتبه‌ی آن در eax بارگذاری می‌شود.

با توجه به اندازه‌گیری زمان دسترسی به حافظه نهان توسط امکانات موجود در پردازنده‌ها، می‌توان گفت حافظه‌ی نهان در حملات کانال جانبی زمانی چالش‌برانگیزتر و کاربردی‌تر است. بر اساس میزان دسترسی مهاجم، حملات کانال جانبی حافظه نهان را می‌توان به دودسته حملات زمان‌محور[2] و حملات مبتنی بر دسترسی به حافظه[3] دسته‌بندی نمود. در حملات زمان‌محور، مهاجم صرفاً توانایی رصد کل زمان اجرای الگوریتم رمزنگاری را دارد [۸, ۹, ۲۲]. در مقابل در حملات مبتنی بر دسترسی به حافظه، مهاجم باید توانایی ایجاد تغییرات در حافظه نهان را داشته باشد [۱۱, ۱۲, ۲۳].

۴-۱- حملات زمان‌محور

استخراج اطلاعات حساس توسط حافظه نهان، اولین بار توسط هو [۲۴] انجام شد. کسلی، اشنایر، وانگر و هال [۲۵]، امکان به‌کارگیری حافظه نهان را به‌عنوان منبعی برای اجرای حملات مبتنی بر نرخ برخورد حافظه نهان موردبررسی قراردادند. تأثیر رفتار حافظه نهان در حملات کانال جانبی زمان مبتنی بر جدول مبنا، توسط تی‌سانو، سیتو، سوزاکی و شیگرای [۷] موردبررسی قرار گرفت و اولین نتایج عملی برای اعمال حملات زمان‌محور علیه DES صورت گرفت. برنشتاین [۱۴] با مشاهده‌ی غیرثابت بودن زمان اجرای الگوریتم رمز،

1 Time stamp counter (TSC)
2 Time Driven
3 Access Driven

توانست حمله‌ای را جهت استخراج کلید AES پیاده‌سازی نماید. این حمله بر روی نسخه پیاده‌سازی شده رمز AES در کتابخانه‌ی OpenSSl انجام گرفت. در ادامه اولین نتایج عملی برای حملات حافظه نهان زمان-محور علیه AES در سال‌های ۲۰۰۷-۲۰۰۴ ارائه شد [۲۲, ۲۶] و در سال ۲۰۰۷، یک مدل مقاوم‌سازی رمزهای متقارن علیه حملات ارائه شد [۲۷]. حملات حافظه نهان زمان‌محور به حملات مبتنی بر تصادم‌های[1] داخلی و حملات نمایه شده[2] دسته‌بندی می‌شوند. یکی از مشکلات در اجرای عملی این حملات این است که مهاجم باید توانایی محاسبه زمان اجرای کل رمز را داشته باشد. علاوه بر این معمولا در این نوع حملات تمام بیت‌های کلید استخراج نمی‌گردد و فقط فضای جستوجوی کلید برای مهاجم کاهش پیدا می‌کند.

۴-۱-۱- حملات زمان‌محور حافظه نهان مبتنی بر تصادم‌های داخلی

حمله‌ی مبتنی بر تصادم‌های داخلی برای سیستم‌های رمزی مانند الگوریتم AES که پیاده‌سازی آنها مبتنی بر جدول مبنا است قابلیت اجرایی دارد. به دلیل قرار داشتن جدول مبنا در قالب‌های حافظه، مهاجم می‌تواند با انتخاب ورودی‌های تصادفی به الگوریتم، آنها را فراخوانی کرده و زمان اجرا را اندازه‌گیری کند. در دومین دسترسی به جدول، زمان اجرای بیشتر، احتمالاً به دلیل عدم حضور بلوک‌های جدول در حافظه نهان و فراخوانی آن از حافظه اصلی بوده که منجر به رخداد فقدان حافظه نهان می‌شود. در حالی که زمان اجرای کمتر به دلیل حضور بلوک‌های جدول در حافظه نهان و رخداد برخورد حافظه نهان است.

در دور اول الگوریتم AES، $s_i = x_i \oplus k_i$ که در آن x_i ، i امین بایت متن آشکار ورودی و k_i امین بایت کلید است. برای تعیین یک تصادم بین s_a و s_b ($1 \le a < b \le 16$) که بر اساس دسترسیِ جدول به‌دست‌آمده است، مهاجم x_a را ثابت نگه می‌دارد (که منجر به ثابت ماندن s_a می‌شود) و برای هر بلوک حافظه از جدول که به‌وسیله‌ی $s_b = x_b \oplus k_b$ قابل‌دسترس است، میانگین زمان اجرا را به دست می‌آورد. البته میانگین زمانی اجرایِ محاسبه‌شده، فقط تصادم متناظر با s_a است. این زمانِ اجرا به‌طور مجزا، از زمان‌های دیگر اجرا متفاوت خواهد بود. تصادم، بیت‌هایی از $\langle k_a \oplus k_b \rangle$ را طبق روابط (۳) آشکار می‌کند:

$$\langle x_a \oplus k_a \rangle = \langle s_b \rangle \Rightarrow \langle x_a \oplus k_a \rangle = \langle x_b \oplus k_b \rangle \Rightarrow \langle k_a \oplus k_b \rangle = \langle x_a \oplus x_b \rangle \quad (۳)$$

که در آن نماد **<>** مقدار بیت‌های پرارزش $k_a \oplus k_b$ را نشان می‌دهد. هر جدول مبنا در پیاده‌سازی AES، ۲۵۶ ($N = 256$) عنصر دارد و هر عنصر ۴ بایت ($B = 4$) را اشغال می‌کند. روی یک سیستم با ۶۴ بایت بلوک حافظه ($L = 64$)، جدول باید در ۱۶ بلوک حافظه قرار بگیرد و هر بلوک حافظه شامل ۱۶ عنصر (یا ۴ بایت) است. کانال جانبی زمانی، ضرورتاً یک تصادم در یک بلوک حافظه را شناسایی می‌کند. تصادم، ۴ بیت را آشکار خواهد کرد ($\log_2(L/B) = \log_2(64/4) = 4$). کلیدهای k_a و k_b برحسب بایت بوده و یک آنتروپی ترکیب‌شده از آن شامل ۱۶ بیت است. حمله مبتنی بر تصادم‌های داخلی، این آنتروپی را به $16 - 4 = 12$ بیت کاهش می‌دهد (درواقع $\langle x_a \oplus x_b \rangle$ همان مقدار آشکارشده‌ی ۴ بیتی $\langle k_a \oplus k_b \rangle$ است). با توجه به این‌که هر جدول مبنای مورداستفاده در اولین دور از AES، چهار مرتبه مورد دسترسی قرار می‌گیرد و هر جدول با جدول دیگر نمی‌تواند تصادم داشته باشد، لذا آنتروپی ۱۲۸ بیتی کلید AES به ۸۰ بیت کاهش پیدا می‌کند.

در مثال فوق به دلیل حذف نویز جدول را چندین بار فراخوانی می‌کنیم. با توجه به این‌که چند هسته در پردازنده به‌طور همزمان در یک سیستم کار می‌کنند، می‌توانند در زمان اجرا تأثیرگذار باشند، درنتیجه نویز به اندازه‌گیری زمان اضافه می‌شود. این نویز می‌تواند به خاطر چند فاکتور دیگر مانند: وقفه‌ها[3]، دسترسی مستقیم به حافظه[4]، دستورالعمل فقدان حافظه نهان، فعال بودن هسته‌های دیگر پردازنده و ... باشد. نویز را می‌توان به‌وسیله‌ی چندین بار فراخوانی جدول با مقادیر یکسان از ورودی و بعد با محاسبه میانگین زمان اجرا، به‌طور قابل‌ملاحظه‌ای کاهش داد. اما همچنین نویز یکنواخت نیست و می‌تواند روی بعضی از مقادیر ورودی نسبت به بعضی دیگر (در اندازه‌گیری زمان) تأثیر بیشتری بگذارد. برای اجتناب از این موضوع، در فراخوانی‌های جدول، مقادیر ورودی به‌طور تصادفی انتخاب می‌شوند.

۴-۱-۲- حملات زمان‌محور حافظه نهان نمایه شده

در سال ۲۰۰۵ برنشتاین [۱۴] با استفاده از حملات زمانی حافظه نهان نمایه‌شده توانست بیت‌هایی از کلید مخفی الگوریتم AES را بازیابی نماید. حملات زمانی حافظه نهان نمایه‌شده از دو فاز تشکیل شده است: یک فاز نمایه‌سازی و در ادامه فاز حمله. در فاز نمایه‌سازی، مهاجم زمان اجرای الگوریتم رمز را با استفاده از کلید معلوم به دست آورده و مشخصه‌های آماری آن را در یک نمایه (که به آن الگو

1 Collision
2 Profiling
3 Interrupt
4 Direct memory access

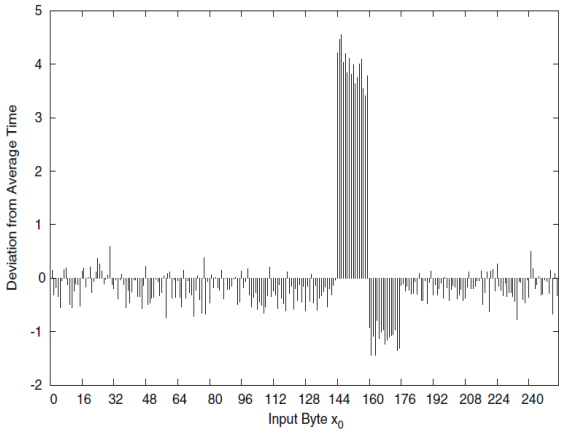
می‌گویند) ذخیره می‌کند. در فاز حمله، نمایه‌ی دیگری برای کلید مخفی ساخته می‌شود. به‌عبارت‌دیگر، مهاجم زمان اجرای الگوریتم رمز را با استفاده از کلیدهای نامعلوم به دست آورده و مشخصه‌های آماری را در یک نمایه‌ی دیگر ذخیره می‌نماید. درنهایت با استفاده از مقایسه آماری بین این دو نمایه‌ی زمانی، یکی از کلیدهای حدسی به‌عنوان کلید مخفی صحیح، آشکار خواهد شد.

به‌عنوان نمونه حملات زمانی حافظه نهان نمایه‌شده را علیه الگوریتم AES بررسی می‌کنیم. با توجه به این‌که AES شانزده بایت ورودی دارد، برای ساختن یک نمایه‌ی زمانی، در ابتدا ورودی‌ها را به‌صورت تصادفی انتخاب می‌کنیم ($X=\left(x_0\|...\|x_i\|...\|x_{15}\right), 0\le i\le 15$)، AES را اجرا و زمان را اندازه‌گیری می‌کنیم. این کار را چندین بار با ورودی‌های مختلف انجام می‌دهیم. ۱۶ آرایه به‌صورت $A_i\left(0\le i\le 15\right)$ تعریف می‌کنیم که اندازه‌ی هرکدام ۲۵۶ بیت است و میانگین زمانی اجرا را در خود ذخیره می‌کنند. عنصر $A_i[j]$ در یک آرایه، میانگین زمانی اجرا را وقتی‌که بایت i امِ متن آشکار برابر j است، در خود ذخیره می‌کند. سپس میانگین زمانی کل هر بایت (مثلاً بایت iام) را به ازای مقادیر مختلفی که می‌تواند بگیرد ($0\le j\le 255$) می‌یابیم (t_{avg}). انحراف معیار را برای هر بایت ورودی به ازای مقادیر مختلفی که می‌تواند بگیرد ($0\le j\le 255$) طبق رابطه‌ی (۴) محاسبه می‌کنیم.

(۴)

$$D_i[j]=A_i[j]-t_{avg},\quad 0\le i\le 15,\quad 0\le j\le 255$$

iامین بایت آرایه‌ی انحراف معیار (D_i) را نمایه‌ی زمانی برای iامین بایت ورودی می‌نامند. بنابراین ۱۶ آرایه‌ی انحراف معیار داریم (یعنی: $\left(D_0, D_1, ..., D_{15}\right)$)، که هرکدام برای هر بایت ورودی است. به‌عنوان‌مثال، نمایه‌ی زمانی محاسبه‌شده برای ورودی x_0 (یا انحراف معیار D_0)، با استفاده از پردازنده‌ی Intel Core2 Duo(E7500)، به‌صورت نمودار (۳) است.



شکل (۳): نمایه زمانی محاسبه‌شده برای اولین ورودی ماتریس حالت [۱۷]

در پیاده‌سازی AES، iامین بایت ورودی با iامین جدول مبنا که در دور اول مورد دسترسی قرارگرفته، متناظر است که با رابطه‌ی $s_i^{(0)}=x_i\oplus k_i^{(0)}$ نشان داده می‌شود. به‌طور مثال برای محاسبه انحراف معیار زمانی (شکل ۳) به ازای مقادیر مختلف ورودی x_0، مقادیر x_0 را ثابت درنظر گرفته و مقادیر سایر x_iها به‌صورت تصادفی مقداردهی می‌شوند. در این‌صورت هنگام اجرای AES، مقدار $s_0^{(0)}$ و بلوک همراهش از جدول مبنا به‌صورت ثابت در کل رمز مورد دسترسی قرار گرفته ولی سایر مقادیر $s_i^{(0)}$ به‌صورت تصادفی مورد دسترسی قرار می‌گیرند. وقتی میانگین زمانی چند میلیون عمل رمزنگاری محاسبه شود نفوذ دسترسی به جدول‌های دیگر در زمان اجرا بی‌تأثیر شده و فقط دسترسی به همان جدول ثابت ($s_0^{(0)}$) در زمان اجرا تأثیرگذار است.

در این حمله برای آشکارسازی iامین بایت کلید مخفی، دو نمایه می‌سازیم: یک نمایه با استفاده از کلیدهای معلوم ($D_i\left[x\oplus\hat{k}_i^{(0)}\right]$) که در آن $\hat{k}_i^{(0)}$ کلید معلوم ثابت برای بایت iام در دور اول و دیگری نمایه‌ای است که با استفاده از کلیدهای حدسی ($D_i\left[x\oplus kguess\right]$) ساخته می‌شود که در آن $kguess$ کلید حدسی ثابت است و در هر دو نمایه x مقادیر ممکن بین 0 تا 255، برای iامین بایت ورودی (یا D_i) می‌باشد. طبق رابطه‌ی (۵)، ضریب همبستگی[1] این دو نمایه را محاسبه می‌کنیم.

(۵)

$$CC_{kguess}=\sum_{x=0}^{255} D_i\left[x\oplus\hat{k}_i^{(0)}\right]\times D_i\left[x\oplus kguess\right]$$

چون کلید حدسی ۲۵۶ حالت دارد، بنابراین به همین تعداد ضریب همبستگی (CC_{kguess}) بین نمایه‌های زمانی تولیدشده به‌وسیله‌ی کلید معلوم و کلید حدسی حاصل می‌شود. درواقع کلید حدسی متناظر با همان کلید مجهول بوده درصورتی‌که ضریب همبستگی، ماکزیمم مقدار را داشته باشد.

۴-۲- حملات مبتنی بر دسترسی به حافظه

با توجه به این‌که نشت اطلاعات در حملات مبتنی بر دسترسی به حافظه نسبت به حملات زمان‌محور بیشتر است [۲۸]، این حملات با استقبال قابل‌توجهی از سوی مهاجمین و جامعه رمزنگاری روبرو شد. در حملات حافظه نهان مبتنی بر دسترسی به حافظه، مهاجم باید به حافظه‌ی نهان قربانی دسترسی داشته باشد. پرسیوال و اسویک [۱۱] به ترتیب، پیشگام در حملات مبتنی بر دسترسی به حافظه بر روی رمزهای RSA و AES بودند. پرسیوال توانست زمان فراخوانی تمام دسته‌های حافظه نهان را اندازه‌گیری نماید و با استفاده از زمان

1 Correlation coefficient

فراخوانی‌شده، مهاجم می‌تواند تخمین بزند که کدام دسته‌ی حافظه نهان توسط قربانی اشغال شده است. اسویک با استفاده از پیشنهادی که پرسیوال مطرح کرده بود، دو روش ارائه داد. اولین روش Evict+Time و دومین روش Prime+Probe نام دارد.

هرچند نتایج حاصل از حملات ارائه‌شده نسبت به حملات پیشین بسیار بهتر می‌باشند، میزان عملیاتی بودن این حملات به‌طور دقیق مشخص نیست. همان‌طور که ذکر شد در حملات مبتنی بر دسترسی به حافظه، فرض می‌شود که مهاجم توانایی ایجاد تغییرات در حافظه نهان را دارد. این فرض سبب شده است که کاربرد حملات مبتنی بر دسترسی به حافظه با محدودیت‌هایی مواجه شود. در همین راستا گولاسچ [۱۶] یک حمله‌ی خیلی قوی ارائه داد و با بهره‌گیری از این حقیقت که اگر مهاجم و قربانی اطلاعات مشترکی را روی خط حافظه نهان قرار دهند، مهاجم می‌تواند حمله‌ی خود را روی سطح L1 حافظه نهان انجام دهد.

یاروم [۱۵] حمله‌ی گولاسچ را توسعه داد و حمله جدیدی به نام Flush+Reload بر روی RSA ارائه نمود که مبتنی بر مشخصه‌های به اشتراک گذاشته‌شده در لایه‌ی L3 حافظه نهان است و مهاجم بدون دسترسی به هسته پردازنده که توسط کاربر به کار گرفته می‌شود، می‌تواند تغییراتی در حافظه‌ی نهان ایجاد نماید. این حمله به‌طور گسترده موردتوجه سایر محققین قرار گرفت. ایده حمله Flush+Reload بر روی AES، ابتدا توسط گولاچ، بانگرتر و کرن [۱۶] معرفی شد. سپس حمله Flush+Reload علیه دور آخر AES انجام و با ۴۰۰۰۰۰ بار عمل رمزنگاری، تمام ۱۲۸ بیت کلید استخراج شد [۳].

حملات مبتنی بر دسترسی به حافظه با استفاده از روش‌های Prime+Prob، Evict+Time، Flush+Reload و Flush+Flush انجام می‌شوند. در این روش‌ها مهاجم با استفاده از ویژگی‌های موجود در معماری پردازنده‌ی اینتل می‌تواند سناریوهای حمله را با استفاده از روش‌های فوق طراحی و پیاده‌سازی نمایند. این ویژگی‌ها شامل موارد زیر است:

- هسته‌های پردازنده اینتل به دلایل زیر می‌توانند به داده‌های یکدیگر دسترسی داشته باشند.
  - شامل بودن اطلاعات لایه‌ی آخر حافظه نهان با اطلاعات لایه‌های بالایی
  - مشترک بودن لایه‌ی سوم
- پشتیبانی از دستور clflush
- پردازنده اینتل دسترسی غیرمجوزدار به شمارنده‌ی سیکل زمانی مانند rdtsc() را دارد.

۴-۲-۱- بررسی ساختار حملات Prime+Probe و Evict+Time

در این زیربخش روش‌های Prime+Probe و Evict+Time روی رمز AES موردبررسی قرار می‌گیرد، با فرض اینکه رمز AES مبتنی بر جدول مبنا پیاده‌سازی شده است. فرض می‌شود برای اولین دور، جدول مبنا T_j، به ازای مقادیر $s_{i+j}^{(0)} = \left(x_{i+j} \oplus k_{i+j}^{(0)}\right)$ برای $0 \le j \le 3$ و $i \in \{0,4,8,12\}$ مورد دسترسی قرارگرفته که در آن x_{i+j} بایت متن آشکار ورودی و $k_{i+j}^{(0)}$ بایت کلید سفیدسازی متناظر است. همچنین عمل رمزنگاری نیز با یک کلید معلوم و حدسی مانند $\tilde{k}_{i+j}^{(0)}$ انجام می‌شود. سپس مقدار ورودی متناظر در جدول مبنا T_j را می‌توان به‌وسیله‌ی $\tilde{s}_{i+j}^{(0)} = \left(x_{i+j} \oplus \tilde{k}_{i+j}^{(0)}\right)$ محاسبه نمود. اگر کلید حدسی صحیح باشد (یعنی: $k_{i+j}^{(0)} = \tilde{k}_{i+j}^{(0)}$) سپس مقدار ورودی محاسبه‌شده به جدول مبنا نیز صحیح بوده (یعنی: $s_{i+j}^{(0)} = \tilde{s}_{i+j}^{(0)}$) و مقدار $s_{i+j}^{(0)}$ و بلوک همراهش از جدول مبنا مورد دسترسی قرار می‌گیرد و بالعکس.

اوسویک، ترومر و شمیر نشان دادند که یک مهاجم از دو روش برای رصد کردن کانال‌های زمانی در حافظه‌ی نهان برای شناسایی مقادیری از $s_{i+j}^{(0)}$ و $\tilde{s}_{i+j}^{(0)}$ که در آن رابطه‌ی تساوی $\left\langle s_{i+j}^{(0)} \right\rangle = \left\langle \tilde{s}_{i+j}^{(0)} \right\rangle$ برقرار باشد، می‌توان استفاده نمود (نماد <> نشان‌دهنده‌ی باارزش‌ترین بیت‌های s است). این روش‌ها شامل Evict+Time و Prime+Probe است. در این روش‌ها مهاجم باید از آدرس‌های دسته‌ای از حافظه‌ی نهان که جدول مبنا در آن بارگذاری می‌شود آگاهی داشته باشد.

- **حمله‌ی** Evict+Time: برای هر مقدار ممکن از $\left\langle \tilde{s}_{i+j}^{(0)} \right\rangle$، مراحل زیر انجام می‌شود:

۱) عمل رمزنگاری AES به ازای یک ورودی تصادفی متن آشکار مانند x و کلید معلوم و حدسی $\tilde{k}_{i+j}^{(0)}$، یکبار انجام می‌شود.

۲) برای یک مقدار حدسی مانند $\tilde{s}_{i+j}^{(0)}$ از جدول، آدرس دسته‌ای از حافظه‌ی نهان که میزبان $T_j\left[\tilde{s}_{i+j}^{(0)}\right]$ است را تعیین کرده و آن را M می‌نامند.

۳) مهاجم عملیاتی را اجرا می‌کند که در آن همه‌ی داده‌های موجود در M پاک[1] شوند. این عملیات شامل بارگذاری داده‌های تصادفی در آدرس M است.

۴) مجدد عمل رمزنگاری با ورودی تصادفی x و کلید نامعلوم $k_{i+j}^{(0)}$ انجام‌شده و زمان اجرا اندازه‌گیری می‌شود.

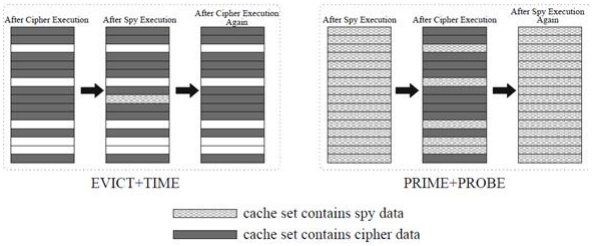
1 Evict

شکل ۱۶ مراحل مختلف حمله و همچنین دسته‌هایی از حافظه‌ی نهان که میزبان جدول مبنا T_j هستند را نشان می‌دهد. بعد از رمزنگاری دوم، دو نتیجه ممکن است اتفاق بیفتد:

- اگر مهاجم زمان بیشتری را رصد نماید در این‌صورت تساوی $\left\langle s_{i+j}^{(0)} \right\rangle = \left\langle \tilde{s}_{i+j}^{(0)} \right\rangle$ برقرار است و دلالت بر آن دارد که $\left\langle k_{i+j}^{(0)} \right\rangle = \left\langle \tilde{k}_{i+j}^{(0)} \right\rangle$. بنابراین در این حالت، دسته‌ی حافظه‌ی نهان M همیشه در دوبار عمل رمزنگاری مورداستفاده قرارگرفته است (در مرحله‌ی ۱ و ۴). در عمل، دومین رمزنگاری همیشه (با احتمال ۱) نتیجه‌ی عدم برخورد حافظه نهان را برای دسترسی $T_j\left[s_{i+j}^{(0)}\right]$ خواهد داشت.
- اگر مهاجم زمان کمتری را رصد کند در این‌صورت نامساوی $\left\langle s_{i+j}^{(0)} \right\rangle \neq \left\langle \tilde{s}_{i+j}^{(0)} \right\rangle$ برقرار است و دلالت بر آن دارد که $\left\langle k_{i+j}^{(0)} \right\rangle \neq \left\langle \tilde{k}_{i+j}^{(0)} \right\rangle$. بنابراین برای یک مقدار از x، $T_j\left[\tilde{s}_{i+j}^{(0)}\right]$ در دسته‌ی حافظه‌ی نهان M قرار نمی‌گیرد. بااین‌حال امکان دارد این دسته از حافظه‌ی نهان میزبان مقادیر دیگر T_j باشد. همچنین احتمال عدم برخورد برای دسترسی $T_j\left[s_{i+j}^{(0)}\right]$ در مرحله‌ی ۴، کوچکتر مساوی یک است.

- **حمله‌ی Prime+Probe**: در روش Evict+Time، زمان دسترسی $T_j\left[s_{i+j}^{(0)}\right]$ معادل کل زمان اجرای رمز است. بااین‌حال ایرادهایی از قبیل وجود دسترسی‌های اضافی به حافظه و کدهایی که همزمان با رمز اجرا می‌شوند، می‌توانند موفقیت این روش را کاهش دهند. بنابراین نویز قابل‌ملاحظه‌ای از منابعی مانند: دستورات زمان‌بندی در پردازنده[1] و وجود انشعاب[2] به وجود می‌آید که باعث می‌شود نسبت سیگنال به نویز (SNR) کاهش یابد. در روش Prime+Probe، کدهای کوچک‌تر (مرتبط با این حمله) زمان‌بندی می‌شوند، درنتیجه حمله از موفقیت بیشتری برخوردار است. مراحل این حمله روی رمز AES به شرح ذیل می‌باشد:

۱) یک آرایه‌ای مانند A و هم‌اندازه با حافظه‌ی نهان تعریف کرده و با فراخوانی مقادیر A، کل حافظه‌ی نهان را پر می‌شود.
۲) عمل رمزنگاری AES به ازای یک ورودی تصادفی مانند x و کلید معلوم و حدسی $\tilde{k}_{i+j}^{(0)}$، یکبار انجام می‌شود.
۳) مهاجم برای یک مقدار حدسی مانند $\tilde{s}_{i+j}^{(0)}$ از جدول مبنا، آدرس دسته‌ای از حافظه‌ی نهان که میزبان $T_j\left[\tilde{s}_{i+j}^{(0)}\right]$ است را تعیین کرده و آن را M می‌نامد.
۴) مهاجم مقادیری از A را که در دسته‌ی M بارگذاری می‌شوند، مجدد فراخوانی کرده و زمان را اندازه‌گیری می‌نماید. (شکل ۴).



شکل (۴): دسته های حافظه نهان متناظر با جدول مبنا T_j برای حملات Evict+Time و Prime+Probe

اگر $\left\langle \tilde{s}_{i+j}^{(0)} \right\rangle$ صحیح باشد (یعنی: $\left\langle s_{i+j}^{(0)} \right\rangle = \left\langle \tilde{s}_{i+j}^{(0)} \right\rangle$)، مقدار A موجود در دسته M هنگام عمل رمزنگاری در مرحله‌ی ۲ (با احتمال ۱) پاک خواهد شد. بااین‌حال، این احتمال می‌تواند از یک کمتر باشد اگر $\left\langle \tilde{s}_{i+j}^{(0)} \right\rangle$ ناصحیح باشد. بنابراین اگر مهاجم زمان طولانی‌تری را رصد نماید به معنای آن است که مقدار A در M (در مرحله ۲ حتماً اتفاق افتاده) پاک شده و در مرحله‌ی چهارم عدم‌برخورد حافظه نهان رخ خواهد داد. بااین‌حال اگر مقدار A در M پاک نشود زمان دسترسی کوتاه‌تر بوده و در مرحله‌ی چهارم یک برخورد حافظه نهان رخ خواهد داد. برای یافتن مقدار صحیح $\left\langle \tilde{s}_{i+j}^{(0)} \right\rangle$ (و درنتیجه مقدار صحیح $\left\langle \tilde{k}_{i+j}^{(0)} \right\rangle$) چهار مرحله‌ی فوق با ورودی‌های مختلف تکرار می‌شود.

۴-۲-۲- بررسی ساختار حملات Prime+Probe و Evict+Time

در این زیربخش روش‌های Flush+Reload و Flush+Flush موردبررسی قرار می‌گیرند.

- **حمله‌ی Flush+Reload**

همان‌طور که اشاره شد، حمله‌ی Flush+Reload مبتنی بر مشخصه‌های به اشتراک گذاشته‌شده در حافظه نهان است. مهاجم بدون دسترسی به هسته پردازنده که توسط کاربر به‌کار گرفته می‌شود، می‌تواند تغییراتی در حافظه‌ی نهان ایجاد کند. حمله‌ی Flush+Reload، شامل چهار مرحله است:

1 Instruction scheduling
2 Conditional branch

مرحله‌ی اول: نگاشت یک منبع به اشتراک گذاشته‌شده در حافظه نهان (این منبع را قربانی و مهاجم هر دو دارند).

مرحله‌ی دوم: مهاجم خط به اشتراک گذاشته‌شده در حافظه نهان را خالی یا اصطلاحاً تخلیه[1] می‌کند (با استفاده از دستور clflush).

مرحله‌ی سوم: در این مرحله مهاجم منتظر می‌ماند تا قربانی از مکان‌هایی از حافظه که در مرحله‌ی قبل تخلیه شده، استفاده کند. فرض بر این است که قربانی داده‌های مربوط به رمزنگاری را بارگذاری کرده است.

مرحله‌ی چهارم: مهاجم داده‌ی به اشتراک گذاشته‌شده در مرحله اول را دوباره بارگذاری کرده و زمان دسترسی به داده را اندازه‌گیری می‌کند.

درصورتی‌که دسترسی مهاجم به داده در مرحله چهارم سریع باشد بدین معنی است، داده موردنظر که در مرحله‌ی دوم تخلیه شده بود، توسط قربانی در مرحله سوم دوباره فراخوانی شده و در حافظه نهان بارگذاری می‌شود. بنابراین در مرحله چهارم مهاجم با اندازه‌گیری، زمان کمتری از بارگذاری را نسبت به زمانی که داده از حافظه اصلی مورد دسترسی قرار می‌گیرد، مشاهده خواهد کرد و برخورد حافظه نهان رخ داده است. درصورتی‌که دسترسی مهاجم به داده در مرحله چهارم سریع نباشد، بدین معنی است که قربانی داده‌ی تخلیه‌شده در مرحله‌ی اول را مورد دسترسی قرار نداده است و بنابراین داده در حافظه نهان موجود نیست و درنتیجه هنگام بارگذاری از حافظه اصلی به حافظه نهان (در مرحله چهارم)، مهاجم بالاترین زمان بارگذاری را در این مرحله مشاهده می‌کند.

- **حمله‌ی Flush+Flush**

حمله Flush+Flush [۱۷] از این حقیقت بهره می‌برد که در صورت عدم انتقال اطلاعات به حافظه نهان، زمان اجرا برای ساختار clflush کم‌تر است و در صورت انتقال اطلاعات به حافظه نهان، زمان اجرا برای ساختار clflush بیشتر است. این حمله نیز روشی برای یافتن خطی از حافظه نهان است که به دلیل اجرای الگوریتم رمز مورد دسترسی قرار می‌گیرد. در هر بار استفاده از دستور clflush، اطلاعات مربوط به تمام سطوح حافظه نهان تخلیه می‌شود. حمله Flush+Flush از ویژگی‌های نرم‌افزاری و سخت‌افزاری حمله Flush+Reload مانند به‌کارگیری حافظه به اشتراک گذاشته‌شده در سطوح مختلف حافظه‌ی نهان و یا در محیط‌های مجازی استفاده می‌کند. برخلاف حمله Flush+Reload، حمله Flush+Flush هیچ دسترسی به حافظه ندارد و بنابراین هیچ فقدان حافظه نهانی رخ نداده و تنها تعداد محدودی برخورد حافظه نهان رخ می‌دهد. به‌طور خلاصه حمله Flush+Flush شامل مراحل زیر است:

مرحله‌ی اول: نگاشت یک منبع به اشتراک گذاشته‌شده در حافظه نهان (این منبع را قربانی و مهاجم هر دو دارند).

مرحله‌ی دوم: مهاجم خط به اشتراک گذاشته‌شده حافظه نهان را تخلیه می‌کند (با استفاده از دستور clflush).

مرحله‌ی سوم: در این مرحله مهاجم منتظر می‌ماند تا قربانی از مکان‌هایی از حافظه که در مرحله‌ی قبلی تخلیه شده، استفاده نماید. فرض بر این است که قربانی داده‌های مربوط به رمزنگاری را بارگذاری کرده است.

مرحله‌ی چهارم: مهاجم خط حافظه نهان به اشتراک‌گذاشته شده را تخلیه کرده و زمان را اندازه‌گیری می‌نماید. مهاجم در خصوص خطی از حافظه نهان که در مرحله‌ی سوم مورد دسترسی قرارگرفته زمان بیشتری را رصد خواهد کرد. به این دلیل که با تخلیه خط حافظه نهان در لایه LLC، با توجه به خاصیت شامل بودن پردازنده، خط حافظه نهان در لایه‌های L1 و L2 نیز تخلیه می‌شوند. مهاجم در خصوص خطی از حافظه نهان که در مرحله‌ی سوم مورد دسترسی قرار نگرفته است زمان کمتری را رصد خواهد کرد، به این دلیل که داده‌ای در خط حافظه نهان قرار ندارد.

۵- نتایج پیاده‌سازی حملات

برای آزمایش سناریوی حملات بخش ۴ به‌صورت عملی و مقایسه نتایج حمله مطابق با جدول ۱، از الگوریتم رمز AES-128 در کتابخانه OpenSSL نسخه 1.1.0 f تحت سیستم‌عامل Ubuntu 16:04 استفاده شده است که روی پردازنده Intel i5-2450M با فرکانس 2.50 GHz پردازش حمله انجام می‌شود.

جدول (۱). مقایسه نتایج پیاده‌سازی حملات

نام حمله	نوع حمله	تعداد نمونه‌گیری‌ها	تعداد بیت‌های استخراج شده کلید
حمله زمان‌محور حافظه نهان مبتنی بر تصادم‌های داخلی روی دور اول AES	متن آشکار انتخابی	2×10^7	۴۸ بیت
حمله زمان‌محور حافظه نهان نمایه شده روی دور اول AES	متن آشکار انتخابی	3×10^8	۶۴ بیت
Evict+Time روی دور اول AES	متن رمزشده انتخابی	4×10^6	۶۴ بیت
Prime+Probe روی دور اول AES	متن آشکار انتخابی	4×10^7	۶۴ بیت
Flush+Reload روی دور آخر AES	متن رمزشده انتخابی	4×10^7	کل بیت‌های کلید
Flush+Flush روی دور آخر AES	متن رمزشده انتخابی	3.5×10^4	کل بیت‌های کلید

1 Flush

در اجرای عملی حملات مشکلات و محدودیت‌هایی وجود دارد که با توجه به جدول ۱، در اجرای دو حمله زمان‌محور باید توانایی محاسبه زمان اجرای کل رمز، برای مهاجم وجود داشته باشد که برای این منظور باید اطلاعات دقیقی از سخت افزار مورد استفاده رمزنگاری در اختیار مهاجم باشد. البته در حملات زمان‌محور بر خلاف حملات مبتنی بر دسترسی به حافظه، مهاجم به دسترسی حافظه نهان قربانی نیازی ندارد؛ و در اجرای چهار حمله بعدی که در دسته‌بندی حملات مبتنی بر دسترسی به حافظه قرار می‌گیرند محدودیت اجرای حمله در ایجاد دسترسی و تعیین خطوط یا دسته‌هایی از حافظه نهان است که این حافظه‌ها توسط رمز مورد دسترسی قرار می‌گیرند. ولی در این حملات به دلیل اینکه مهاجم به دسته‌ها و خطوط حافظه نهان قربانی دسترسی دارد، کل بیت‌های کلید الگوریتم AES را می‌تواند استخراج کند.

در روش‌های Evict+Time و Prime+Probe مهاجم به دسته‌های حافظه نهان و در روش‌های Flush+Reload و Flush+Flush مهاجم به خطوط حافظه نهان دسترسی دارد. همچنین روش Flush+Flush در مقایسه با روش Flush+Reload به دلیل این‌که رخداد عدم فقدان حافظه نهان در آن اتفاق نمی‌افتد، کارایی آن بالاتر بوده و به عنوان برنامه های مخرب پایه[1] قابل شناسایی نیست.

۶- نتیجه

در این مقاله، پس از مرور پیاده‌سازی نرم‌افزاری الگوریتم AES و ساختار حافظه نهان پردازنده اینتل، حملات کانال جانبی زمان مبتنی بر ریزمعماری پردازنده‌ها مورد بررسی قرار گرفت. با توجه به ساختار حافظه‌ی نهان، حملات کانال جانبی زمانی مبتنی بر حافظه نهان کاربردی‌تر بوده و می‌تواند تهدید جدی و عملی علیه سامانه‌های امنیتی نسبت به سایر حملات کانال جانبی زمانی داشته باشد. بنابراین انواع حملات مبتنی بر حافظه نهان مورد بررسی قرار گرفته و به صورت عملی پیاده‌سازی شدند. نتایج پیاده‌سازی نشان می‌دهد که در حملات مبتنی بر دسترسی به حافظه به دلیل اینکه مهاجم به به دسته‌ها و خطوط حافظه نهان قربانی دسترسی دارد، کل بیت‌های کلید الگوریتم AES را می‌تواند استخراج کند ولی در حملات زمان‌محور، مهاجم به دسترسی حافظه نهان قربانی نیازی ندارد و با اطلاعات کمتری از قربانی مواجه است.

در ادامه کار پیشنهاد می‌گردد اجرای حملات کانال جانبی زمان بر روی الگوریتم‌های رمزهای جریانی[2] مورد بررسی قرار گرفته و نتایج اجرای این حملات بر روی یک نمونه از الگوریتم‌های رمز جریانی، مورد مقایسه با اجرای حملات بر روی رمز AES قرار گیرند.

[1] Malicious based
[2] Stream

سپاسگزاری

از حمایت‌های مرکز فتح دانشگاه جامع امام حسین (ع) کمال سپاسگزاری را داریم.

مراجع

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

[2] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in International Conference on Cryptographic Hardware and Embedded Systems, 2016: Springer, pp. 368-388 .

[3] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! A fast, Cross-VM attack on AES," in International Workshop on Recent Advances in Intrusion Detection, 2014: Springer, pp. 299-319 .

[4] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache Attacks on Mobile Devices," in USENIX Security Symposium, 2016, pp. 549-564 .

[5] Y. Yarom and N. Benger, "Recovering OpenSSL ECDSA Nonces Using the FLUSH+ RELOAD Cache Side-channel Attack," IACR Cryptology ePrint Archive, vol. 2014, p. 140, 2014.

[6] A. Canteaut, C. Lauradoux, and A. Seznec, "Understanding cache attacks," INRIA, 2006 .

[7] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, "Cryptanalysis of DES implemented on computers with cache," in International Workshop on Cryptographic Hardware and Embedded Systems, 2003: Springer, pp. 62-76 .

[8] J. Bonneau and I. Mironov, "Cache-collision timing attacks against AES," in International Workshop on Cryptographic Hardware and Embedded Systems, 2006: Springer, pp. 201-215 .

[9] H. Aly and M. ElGayyar, "Attacking aes using bernstein's attack on modern processors," in International Conference on Cryptology in Africa :۲۰۱۳ ,Springer, pp. 127-139 .

[10] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in Cryptographers' Track at the RSA Conference, 2006: Springer, pp. 1-20 .

[11] C. Percival, "Cache missing for fun and profit," ed: BSDCan, 2005.

[12] M. Neve and J.-P. Seifert, "Advances on access-driven cache attacks on AES," in International Workshop on Selected Areas in Cryptography, 2006: Springer, pp. 147-162 .

[13] C. Rebeiro, D. Mukhopadhyay, and S. Bhattacharya, Timing Channels in Cryptography: A Micro-Architectural Perspective. Springer, 2014.

[14] D. J. Bernstein, "Cache-timing attacks on AES," ed: Citeseer, 2005.

[15] Y. Yarom and K. Falkner, "FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in USENIX Security Symposium, 2014, pp. 719-732 .

[16] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games--bringing access-based cache attacks on AES to practice," in Security and Privacy (SP), 2011 IEEE Symposium on, 2011: IEEE, pp. 490-505 .

[17] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ Flush: a fast and stealthy cache attack," in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2016: Springer, pp. 279-299 .

[18]]J. Daemen and V. Rijmen, The design of Rijndael: AES-the advanced encryption standard. Springer Science & Business Media, 2013.

[19] B. B. Brumley, Covert timing channels, caching, and cryptography. Aalto University, 2011.

[20] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: a timing attack on OpenSSL constant-time RSA," Journal of Cryptographic Engineering, vol. 7, no. 2, pp. 99-112, 2017.

[21]]Q. Ge, Y. Yarom, F. Li, and G. Heiser, "Contemporary Processors Are Leaky–and There's Nothing You Can Do About It," The Computing Research Repository. arXiv, 2016.

[22] M. Neve, J.-P. Seifert, and Z. Wang, "A refined look at Bernstein's AES side-channel analysis," in Proceedings of the 2006 ACM Symposium on Information, computer and communications security, 2006, pp. 369-369 .

[23] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient cache attacks on AES, and countermeasures," Journal of Cryptology, vol. 23, no. 1, pp. 37-71, 2010.

[24] W.-M. Hu, "Lattice scheduling and covert channels," in Proceedings 1992 IEEE Computer Society Symposium on Research in Security and Privacy, 1992: IEEE Computer Society, pp. 52-5 .۲

[25]]J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers," in European Symposium on Research in Computer Security, 1998: Springer, pp. 97-110 .

[26] O. Acıiçmez, W. Schindler, and Ç. K. Koç, "Cache based remote timing attack on the AES," in Cryptographers' track at the RSA conference, 2007: Springer, pp. 271-286 .

[27] K. Tiri, O. Acıiçmez, M. Neve, and F. Andersen, "An analytical model for time-driven cache attacks," in International Workshop on Fast Software Encryption, 2007: Springer, pp. 399-413 .

[28] C. Maurice, "Information leakage on shared hardware: evolutions in recent hardware and applications to virtualization," 2015 .