علوم و مهندسی کامپیوتر
نشریه علمی پژوهشی انجمن کامپیوتر ایران
جلد ۴ شماره ۳ (الف)، ۱۳۸۵
صفحات ۲۷-۳۶
مقاله کوتاه

نشریه علمی پژوهشی انجمن کامپیوتر ایران

تحلیل زمانی الگوریتم ضرب‌پیمانه‌ای Blakley

بهادر بخشی بابک صادقیان

دانشکده مهندسی کامپیوتر و فناوری اطلاعات، دانشگاه صنعتی امیر کبیر، تهران، ایران

چکیده

حمله‌های زمانی منتشر شده بر روی توان‌رسانی پیمانه‌ای، مبتنی بر تغییرات زیاد در زمان اجرای آنهاست و چنین فرض می‌شود که زمان اجرای هر مرحله از الگوریتم مستقل از زمان اجرای سایر مراحل است. در این مقاله طرح حمله جدیدی بر روی ضرب پیمانه‌ای Blakley که تغییرات زمان اجرای آن به مراتب کمتر از تغییرات زمان اجرای توان‌رسانی پیمانه‌ای است ارایه می‌شود. نشان می‌دهیم که فرض استقلال زمان اجرای مراحل مختلف الگوریتم در الگوریتم Blakley معتبر نیست. یک مدل‌سازی ریاضی از این همبستگی ارایه می‌کنیم. فرض می‌کنیم که یک مجموعه از اعداد که حمله کننده می‌داند با یک پارامتر یکسان مخفی ضرب می‌شود و زمان اجرای الگوریتم ضرب پیمانه‌ای برای هر ضرب داده شده است، اما نتیجه عمل ضرب در دست نیست. همچنین ماشین مشابه ماشین تحت حمله را در اختیار نداریم. در برخی از کاربردها مانند استاندارد امضای دیجیتال، این پارامتر مخفی کلید محرمانه است. با استفاده از همبستگی‌های بدست آمده، طرح حمله زمانی را برای یافتن پارامتر مخفی که همان کلید محرمانه است ارایه می‌کنیم. علاوه بر طرح حمله روش تشخیص خطایی ارایه می‌شود. همچنین روشی تصحیح خطا برای افزایش کارایی حمله پیشنهاد می‌کنیم. پیاده‌سازی عملی این طرح حمله بر روی DSS نشان می‌دهد که احتمال خطا کمتر از ۰/۱۵ است و کلید مخفی ۱۶۰ بیتی با استفاده از ۱،۵۰۰،۰۰۰ نمونه زمانی بدست می‌آید.

کلمات‌کلیدی: حمله زمانی، ضرب پیمانه‌ای، الگوریتم Blakley، همبستگی

۱- مقدمه

تا دهه اخیر تمام تحلیل‌های الگوریتمهای رمز بر اساس ویژگیهای ریاضی الگوریتم استوار بودند، یکی از معروفترین نمونه از این نوع حملات، حمله تفاضلی است. در دهه اخیر نوع جدیدی از حملات مطرح شده است که به آن Side-Channel Attack می‌گویند[۱]. ایده اصلی این روش استفاده از اطلاعات نشتی است. وقتی که الگوریتم رمزنگاری به صورت عملی پیاده سازی شود، در حین انجام عملیات رمزنگاری اطلاعاتی مانند مقدار جریان مصرفی، مدت زمان اجرای الگوریتم و ... از سیستم نشت می‌کنند که حمله کننده می‌تواند از آنها استفاده کند. تا کنون کانالهای جانبی متعددی برای این منظور استفاده شده است. که یکی از مهمترین آنها کانال زمان اجرا است. این نوع حملات بخصوص در محیطهای نهفته و کارتهای هوشمند قابل اجرا است. در این کاربردها امکان اندازه‌گیری زمان اجرا الگوریتم وجود دارد.

انجام عملیات رمزنگاری معمولا طولانی و زمان‌بر است، وقتی این الگوریتمها در سیستمی پیاده‌سازی می‌شوند، برای بالا بردن کارایی، بهینه‌سازی‌های زیادی انجام می‌گیرد، برای مثال دستورات شرطی وابسته به مقدار ورودی در این پیاده‌سازی‌ها وجود دارد تا عملیات زایدی انجام نگیرد. این چنین دستوراتی موجب می‌شود که زمان لازم برای رمزکردن یا ترجمه رمز یک متن نه تنها به الگوریتم استفاده شده برای اینکار، بلکه به خود مقدار متن و از همه مهمتر به کلید مخفی رمزنگاری نیز وابسته باشد. Kocher اولین کسی بود که حمله زمانی را در مقاله خود مطرح ساخت. در سال ۱۹۹۶ وی نشان داد که چگونه زمان اجرای الگوریتمهایی مانند RSA به مقدار کلید آنها وابستگی دارد و چگونگی بکارگیری این داده‌ها برای حمله به الگوریتم را نیز مورد بررسی قرار داد[۲]. طرح حمله Kocher، مبتنی بر استفاده از واریانس زمان اجرای هر مرحله از الگوریتم بود.

بعد از مقاله Kocher کارهای متعددی با استفاده از کانال زمان اجرا انجام شد. عمده این کارها بر روی الگوریتم RSA انجام شده است. در [۱] طرح Kocher برای پیاده‌سازی توان‌رسانی که از Sliding-window استفاده می‌کنند گسترش داده شد. Kocher به صورت عملی ایده خود را پیاده‌سازی نکرده و در عمل آنرا

مورد بررسی قرار نداد. اولین نمونه عملی حمله زمانی توسط J.-F Dhem در سال ۱۹۹۸ بر روی RSA ارایه شد[۳]. طرحی که برای حمله استفاده شده بود بهره‌بردن از یک ویژگی سیستم و دسته‌بندی پیام‌ها بر اساس این ویژگی است. این ویژگی به نحوی انتخاب می‌شود که معنی‌دار یا بی‌معنی بودن دسته‌بندی وابسته به کلید مخفی باشد. این حمله‌ها قابل اعمال بر روی پیاده‌سازی‌های RSA که از CRT (Remainder Theorem Chinese's) استفاده می‌کنند، نیست. W. Schindler در سال ۲۰۰۰ این مسئله را حل کرد[۴]. در طرح وی برای حمله به RSA، فرض بر این است که ضربهای پیمانه‌ای با استفاده از الگوریتم Montgomery انجام می‌شود. با محاسبه احتمال رخ دادن تفریق اضافی در الگوریتم Montgomery و رابطه همبستگی بین زمان اجرا و رخ دادن تفریق اضافی وی توانست بیتهای کلید مخفی را بیابد. کار مشابهی با کار Schindler و مستقل از وی توسط Walter انجام شد، وی نیز با تمرکز بر روی تفریق اضافه‌ای که ممکن است در الگوریتم Montgomery انجام شود، این ضرب را بررسی کرده و طرحی برای یافتن توان در الگوریتم RSA با استفاده از همین طرح ارایه کرد[۵]. لازم به ذکر است که الگوریتم Montgomery بعدا توسط Sato در سال 2004 تحلیل شد و احتمال رخ دادن تفریق اضافی به صورت دقیق برای حالتهای ضرب و توان دو محاسبه شد[۶].

علاوه بر الگوریتم RSA، تحلیل زمانی بر روی سایر الگوریتم‌ها نیز اعمال شده است. در ابتدا چنین تصور می‌شد که تنها الگوریتمهایی که از اعمال جبری بر روی اعداد بزرگ استفاده می‌کنند در برابر حمله زمانی ضعیف هستند ولی سه الگوریتم از الگوریتمهای کلید متقارن، DES، RC5 و Rijndeal نیز مورد تحلیل زمانی قرار گرفتند[۷، ۸ و ۹] و نشان داده شد که پیاده‌سازی این‌گونه از الگوریتم‌ها هم به اندازه کافی دارای تغییرات زمانی هستند که بتوان از آن برای حمله زمانی استفاده کرد.

تقریبا در تمام حملاتی که عنوان شد، فرض بر این بود که محیطی که الگوریتم رمز در آنجا می‌شود، محیطی مانند کارت هوشمند است و تحلیلگر با دقت بالایی می‌تواند این زمان‌ها را اندازه‌گیری کند. در سال 2003، Brumley نشان داد که این فرض الزامی نیست. وی طرح حمله‌ای ارائه کرد که توانست به صورت راه دور پیاده‌سازی RSAی که در نرم‌افزار OpenSSL وجود دارد را مورد حمله قرار دهد[۱۰].

یکی از اعمالی که در رمزنگاری‌های کلید عمومی و امضای دیجیتال استفاده زیادی دارد، ضرب پیمانه‌ای است که حاصلضرب دو عدد a و b به پیمانه عددی مانند q محاسبه می‌شود. یکی از رایجترین الگوریتمهای ضرب پیمانه‌ای، الگوریتم interleaved یا همان Blakley است. در این مقاله یک تحلیل زمانی برای این الگوریتم ارایه می‌شود، در حالی که هیچ حمله زمانی دیگری روی این الگوریتم تاکنون گزارش نشده است. در ادامه در بخش دوم الگوریتم ضرب پیمانه‌ای Blakley معرفی می‌شود، در بخش سوم به مسئله همبستگی بین مراحل مختلف الگوریتم و تاثیر آن بر روی حمله زمانی می‌پردازیم سپس در بخش چهارم طرح حمله معرفی می‌شود. نتایج عملی این طرح در بخش پنجم ارایه شده است. روش تشخیص و تصحیح خطا در بخش ششم بررسی خواهد شد.

۲- الگوریتم ضرب پیمانه‌ای Blakley

برای محاسبه ضرب پیمانه‌ای، $a.b \bmod q$، روشهای متعددی ارایه شده است [۱۱] و [۱۲]. الگوریتم Blakley یکی از این روشهاست. دلیل اینکه از الگوریتم Montgomery برای پیاده‌سازی ضرب پیمانه‌ای در تمام الگوریتم‌ها پیشنهاد نمی‌شود نیاز به پیش‌پردازش، تولید پارامتر و پس‌پردازش در الگوریتم Montgomery است. این مراحل هزینه بالایی دارند بنابراین Montgomery برای محاسبه ضربهایی که تنها یکبار اجرا خواهد شد، مانند تولید امضا در DSAمناسب نیست، بلکه کاربرد اصلی آن در محاسباتی مانند توان‌رسانی پیمانه‌ای است که چندین ضرب متوالی در آنها وجود دارد و در مراحل میانی نیازی به پیش/پس‌پردازش وجود ندارد. در این روش، کاهش نتیجه و الگوریتم ضرب باهم انجام می‌شود. فرض بر این است که ورودی‌های الگوریتم کمتر از پیمانه باشند یعنی $a < q$ و $b < q$.

فرض کنیم تمام اعداد t بیتی هستند، برای محاسبه حاصلضرب پیمانه‌ای به صورت زیر عمل می‌کنیم:

$$a = (a_{t-1}a_{t-2}..a_0)$$
$$b = (b_{t-1}b_{t-2}..b_0)$$
$$a.b \bmod q = (\sum_{i=0}^{t-1} 2^i a_i b) \bmod q$$
$$= \sum_{i=0}^{t-1} \left[(2^i a_i b) \bmod q \right]$$
$$= a_0 b + 2(a_1 b + ...2(a_{t-1} b) \bmod q ...) \bmod q$$

بیان الگوریتمی حاصل جمع بالا به این صورت خواهد بود:

ALGORITHM Blakley
INPUT: Three integers a, b and q such that $a < q$ and $b < q$
OUTPUT: Integer c such that $c = a.b \bmod q$
1. $p = 0$
2. for $j = t - 1$ to 0 do
3. $\quad p = p + p$
4. $\quad d = p - q$
5. $\quad$ if $d > 0$ then $p = d$
6. $\quad$ if $(a_j == 1)$ then
7. $\quad\quad p = p + b$
8. $\quad\quad d = p - q$
9. $\quad\quad$ if $(d > 0)$ then $p = d$
10. return $c = p$

مراحل ۴، ۵ و ۸، ۹ کاهش به پیمانه q انجام می‌شود و از آنجا که $p < 2q$ است، همیشه یک تفریق کافی است.

در این پیاده‌سازی دستورات شرطی وابسته به مقدار ورودی دارد. شرط مرحله ششم بر اساس بیتهای ورودی a انجام می‌گیرد و شرطهای مرحله پنجم و نهم وابسته به مقدار p است که از روی ورودی‌های ساخته می‌شود. بنابراین این پیاده‌سازی در برابر حمله زمانی آسیب‌پذیر خواهد بود. زیرا همانطور که Kocher نشان داد، اگر بتوان همبستگی معنی‌داری بین زمان اجرا و تغییرات آن به ازای ورودی‌های مختلف و اجرای مخفی سیستم پیدا کرد، پیاده‌سازی با استفاده از حمله زمانی شکسته خواهد شد.

۳- همبستگی و تاثیر آن بر حمله زمانی

الگوریتم Blakley از یک حلقه تشکیل شده است که عمل اصلی این حلقه جمع پیمانه‌ای است و در نتیجه زمان اجرای کل الگوریتم حاصل جمع زمان اجرای هر یک از جمع‌های پیمانه‌ای است. در برخی از حملات زمانی، زمان اجرای هر یک از مراحل مستقل از سایر مراحل در نظر گرفته می‌شود، مانند طرح حمله Kocher. این فرض باعث می‌شود تا مشخصه‌های آماری زمان اجرای کل (مانند واریانس زمان اجرای کل) را به صورت مجموع مشخصه آماری هر یک از مراحل بیان کرد و همچنین هر بیت کلید را به صورت مستقل از سایر بیت‌ها و بدون در نظر گرفتن

تاثیر آنها حدس زد. در روند انجام الگوریتم، جمع‌های پیمانه‌ایی پشت سرهم انجام می‌شوند که یکی از ورودی‌ها یا هر دو ورودی عمل جمع بعدی، خروجی جمع قبلی است. برای بررسی تاثیر مراحل بر روی هم، همبستگی بین دو عمل جمع متوالی مد نظر خواهد بود.

با تقریب بسیار خوبی می‌توان زمان اجرای جمع پیمانه‌ای را یک متغیر تصادفی برنولی دانست. اگر نتیجه جمع دو عملوند بزرگتر از q باشد، تفریق اضافه‌ای برای کاهش نتیجه به پیمانه q انجام می‌شود و باعث می‌شود تا زمان اجرا افزایش یابد در غیر این صورت زمان اجرا تقریبا مستقل از مقدار ورودی‌ها و برابر مقدار ثابتی خواهد بود. بنابراین انجام شدن/نشدن کاهش اضافی مقدار زمان اجرای جمع پیمانه‌ای را مشخص می‌کند، در نتیجه برای بررسی تاثیر جمع‌های پیمانه‌ای متوالی بر روی همدیگر کافی است تا همبستگی بین اجرا شدن/نشدن کاهش اضافی در جمع‌های متوالی بررسی شود.

برای این بررسی در ادامه از این تعاریف استفاده خواهیم کرد:

- جمع۱: جمع پیمانه‌ای $p + p \bmod q$ که توسط مراحل ۳، ۴ و ۵ از الگوریتم پیاده‌سازی شده است.
- جمع۲: جمع پیمانه‌ای $p + b \bmod q$ که توسط مراحل ۷، ۸ و ۹ از الگوریتم پیاده‌سازی شده است.
- رخ دادن $Er_{1,i}$: اگر شرط مرحله پنجم در دور $j = i$ درست باشد به معنای انجام کاهش اضافی است و آنرا رخ دادن $Er_{1,i}$ می‌نامیم.
- رخ دادن $Er_{2,i}$: اگر شرط مرحله نهم در دور $j = i$ درست باشد آنرا رخ دادن $Er_{2,i}$ می‌نامیم.
- مقدار p در هر دور $j = i$ را با p_i نشان می‌دهیم

با توجه به شرط ورودی‌های الگوریتم ($a < q, b < q$)، کاهش به پیمانه q در هر مرحله و به شرط استفاده از مولد تصادفی با توزیع یکنواخت برای ورودی‌ها، فرض‌های زیر معتبر خواهند بود:

۱- عملوندهای جمع۱ و جمع۲ کوچکتر از q است.

۲- عملوندهای جمع۲ مستقل از هم هستند.

۳- عملوندها در بازه ۰ تا q توزیع یکنواخت دارند و احتمال هر مقدار برابر $1/q$ است

سه حالت برای دو جمع پشت سرهم وجود دارد:

الف)جمع۱ در دور $k+1$ (که در آن $j = t-k-1$ است) بعد از جمع۱ در دور k (که در آن $j = t-k$ است)

ب) جمع۱ در دور $k+1$ بعد از جمع۲ در دور k

ج) جمع۲ در دور k بعد از جمع۱ در دور k

حالت الف: این حالت زمانی رخ می‌دهد که بیت کلید در دور $j = i$ برابر صفر باشد، در این صورت جمع۱ در این دور انجام می‌شود و عمل بعدی که انجام خواهد شد، جمع۱ در دور $j = i-1$ است که هر دو ورودی آن، خروجی جمع۱ در دور $j = i$ است. احتمال رخ دادن کاهشهای اضافی به شرح زیر است:

احتمال رخ دادن $Er_{1,i}$: $Prob(Er_{1,i}) = Prob(p > \frac{q}{2}) = 0.5$.

احتمال رخ دادن $Er_{1,i-1}$ و رخ دادن $Er_{1,i}$:

$$Prob(Er_{1,i-1}, Er_{1,i})$$
$$= Prob((p_{i-1} > \frac{q}{2}) \mid (p_i > \frac{q}{2})).Prob(p_i > \frac{q}{2})$$
$$= Prob(((2p_i - q) > \frac{q}{2}) \mid (p_i > \frac{q}{2})).Prob(p_i > \frac{q}{2})$$
$$= Prob(p_i > \frac{3q}{4}) = 0.25$$

و به صورت مشابه:

احتمال رخ دادن $Er_{1,i-1}$ و رخ ندادن $Er_{1,i}$:

$$Prob(Er_{1,i-1}, \tilde{Er}_{1,i}) = Prob(\frac{q}{2} < p_i < \frac{3q}{4}) = 0.25$$

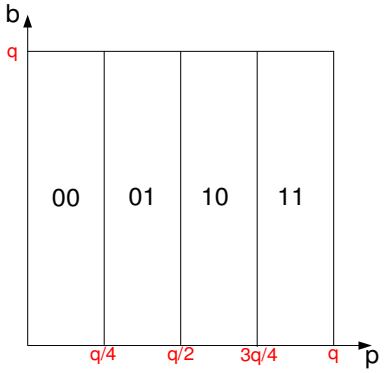
احتمال رخ ندادن $Er_{1,i-1}$ و رخ دادن $Er_{1,i}$:

$$Prob(\tilde{Er}_{1,i-1}, Er_{1,i}) = Prob(\frac{q}{4} < p_i < \frac{q}{2}) = 0.25$$

احتمال رخ ندادن $Er_{1,i+1}$ و رخ ندادن $Er_{1,i}$:

$$Prob(\tilde{Er}_{1,i-1}, \tilde{Er}_{1,i}) = Prob(p_i < \frac{q}{4}) = 0.25$$

توزیع این احتمالها برحسب دو متغیر تصادفی p و b در صفحه ضرب کارتزین این دو متغیر، در شکل ۱ نشان داده شده است.



شکل۱- توزیع احتمال Er برای حالت الف

در این شکل به هر ناحیه صفحه با یک کد xy که به صورت زیر تعریف شده است اشاره می‌شود:

$$xy := \{x = 1 \text{ if } Er_{1,i} \text{ else } x = 0; y = 1 \text{ if } Er_{1,i-1} \text{ else } y = 0\}$$

حالتهای ب و ج: این حالتها زمانی وجود خواهند داشت که بیت کلید در دور $j = i$ یک باشد. مشابه محاسباتی که برای رخ دادن تفریق اضافی در حالت الف انجام شد برای حالت ب و ج تکرار می‌شود با این تفاوت که شرطهای موجود برای p_i در محاسبه احتمالها متفاوت خواهد بود. برای مثال احتمال رخ دادن $Er_{1,i-1}$ به شرط رخ دادن $Er_{2,i}$ در حالت ب برابر است با:

$$Prob(Er_{1,i-1}, Er_{2,i})$$
$$= Prob((p_{i-1} > \frac{q}{2}) \mid (p_i + b > q)).Prob(p_i + b > q)$$
$$= Prob(((p_i + b - q) > \frac{q}{2}) \mid (p_i + b > q)).Prob(p_i + b > q)$$
$$= Prob(2p_i + 2b > 3q) = 0.25$$

نتایج این محاسبات در صفحه p و b برای حالت ب و ج به ترتیب در شکل‌های ۲ و ۳ نشان داده شده است. در شکل ۲ کد xy نسبت داده شده به هر ناحیه براساس دو رخ دادن/ندادن $Er_{1,i-1}$, $Er_{2,i}$ به این صورت تعریف می‌شود:

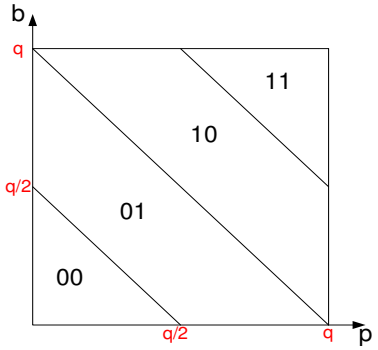
$$xy := \{x = 1 \text{ if } Er_{2,i} \text{ else } x = 0; y = 1 \text{ if } Er_{1,i-1} \text{ else } y = 0\}$$

برای شکل ۳ کدها از این رابطه بدست می‌آید:

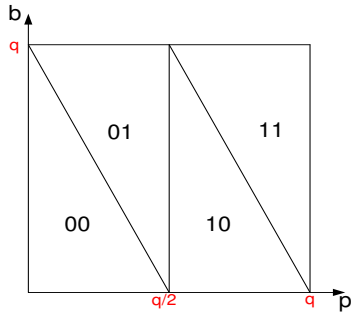
$$xy := \{x = 1 \text{ if } Er_{1,i} \text{ else } x = 0; y = 1 \text{ if } Er_{2,i} \text{ else } y = 0\}$$

با بررسی اشکال ۲، ۳ و ۴، همبستگی زمان اجرای دو عمل ضرب متوالی در الگوریتم ضرب پیمانه‌ای Blakley را آشکار می‌کند. در هر یک از این حالتها اگر در انجام عمل ضرب پیمانه‌ای زوج (p_i, b) در ناحیه 00 یا 11 باشد در این

صورت دو جمع متوالی رفتار زمانی یکسانی از خود نشان می‌دهند (هر دو زمان اجرای کمی دارند یا زمان اجرای هر دو زیاد است) پس ضریب همبستگی بین زمان اجرای این دو مرحله برابر یک خواهد بود. و اگر زوج (p_i, b) در ناحیه 01 یا 10 باشد زمان اجرای دو جمع متوالی مخالف هم‌دیگر رفتار می‌کند و در نتیجه ضریب همبستگی بین این دو مقدار منفی خواهد بود.



شکل۲- توزیع احتمال Er برای حالت ب



شکل۳- توزیع احتمال Er برای حالت ج

بنابراین همانطور که قبلا نیز اشاره شد وجود همبستگی بین زمان اجرای مراحل مختلف باعث خواهد شد که نتوان از طرح‌های حمله Kocher و یا Dhem به صورت مستقیم برای حمله استفاده کرد. در این ادامه نشان داده شده است که چگونه می‌توان با در نظر گرفتن این همبستگی‌ها طرح حمله‌ای مشابه طرح Dhem ارایه کرد که به صورت عملی توانایی شکستن الگوریتم ضرب پیمانه‌ای را دارد.

۴- طرح حمله

در این بخش طرح حمله زمانی برای الگوریتم ضرب Blakley ارایه می‌شود. طرحی‌که در اینجا ارایه شده، براساس طرحی است که در [۳] معرفی شده است. فرضیات حمله عبارتند از:

- از الگوریتم Blakley برای محاسبه $x.r \bmod q$ استفاده می‌شود.
- x کلید مخفی است و به عنوان پارامتر a به الگوریتم داده می‌شود.
- پارامتر عمومی است و حمله کننده مقدار آن را به ازای اجراهای متعدد الگوریتم می‌داند
- حمله کننده می‌تواند زمان اجرای کل الگوریتم بر روی ماشین تحت حمله را اندازه‌گیری کند.
- تحلیل‌ها و نتایج عملی ارایه شده در این مقاله به ازای کلید ۱۶۰ بیتی است که البته اندازه کلید می‌تواند هر مقدار دلخواهی باشد و نحوه تحلیل تغییری نخواهد داشت.

طرحی کلی کار به این صورت است: تحلیلگر با داشتن i بیت اول کلید برای یافتن بیت بعدی، الگوریتم را تا بیت $i+1$ شبیه‌سازی کرده و مجموعه پیامها و مجموعه زمان‌ها را بر اساس یک ویژگی از سیستم به دو زیر مجموعه افراز می‌کند. تابعی بر روی مشخصه‌های آماری (مانند میانگین یا واریانس) دو مجموعه حاصل از افراز مجموعه زمان‌ها تعریف می‌کند، این تابع به نحوی تعریف می‌شود که مقدار آن وابسته به بیت کلید است. تحلیلگر با استفاده از مقدار تابع، این بیت کلید را حدس می‌زند. توصیف رسمی این طرح در[۳] آمده است.

برای حمله زمانی به الگوریتم Blakley، ویژگی سیستم رخ دادن یا رخ ندادن $Er_{2,i}$ انتخاب شده است. مشخصه آماری که از آن استفاده خواهد شد میانگین مجموعه‌های زمان‌ها است. تابعی تصمیم که بر روی میانگین این دو مجموعه تعریف می‌شود تفاضل میان میانگین‌ها است که آنرا با $diff$ نشان می‌دهیم. در ادامه با در نظر گرفتن زمان اجرای الگوریتم طرح حمله با جزییات کامل شرح داده می شود و نشان داده خواهد شد که استفاده از سه پارامتر $Er_{2,i}$، میانگین و تفاضل در طرح حمله، قابلیت کشف بیت‌های کلید مخفی استفاده شده در الگوریتم ضرب Blakley را دارد.

زمان اجرای الگوریتم ضرب برای مقدار r برابر است با:

$$T(r) = \sum_{j=159}^{0} (t_1 + \alpha_j t_2 + \beta_j (t_1 + \gamma_j t_2)).$$

که در آن:

- t_1: زمان اجرای مرحله ۳ و ۴ یا زمان اجرای مراحل ۷ و ۸ است.
- t_2: زمان اجرای مرحله ۵ یا ۹ از الگوریتم است.
- α_j: معادل رخ دادن $Er_{1,j}$ است. اگر شرط مرحله ۵ درست باشد مقدار α برابر ۱ است
- β_j: برابر بیت j-ام x است. اگر برابر ۱ باشد مراحل ۷، ۸ و شاید ۹ اجرا شود.
- γ_j: معادل رخ دادن $Er_{2,j}$ است، اگر شرط مرحله ۹ درست باشد مقدار γ برابر ۱ است.

فرض می‌کنیم که تحلیلگر:

۱- می‌تواند تعداد دلخواهی r را که با x ضرب شده‌اند را جمع آوری کند: $R = \{r_1, r_2, ..., r_k\}$

۲- می‌تواند زمان اجرای کل الگوریتم ضرب را برای هر پیام اندازه‌گیری کند: $T = \{T_1, T_2, ..., T_k\}$ که T_i زمان اجرای الگوریتم به ازای r_i است.

۳- مجموعه $\{\beta_{159}, \beta_{158}, \beta_{157}, ..., \beta_{i+1}\}$ را در اختیار دارد، به عبارت دیگر $159-i$ بیت اول کلید را می‌داند.

هدف یافتن مقدار β_i است. برای شروع حمله فرض می‌کنیم β_i برابر۱ است و برای هر $r_i \in R$ الگوریتم تا مرحله i اجرا می‌شود. با توجه به رخ دادن $Er_{2,i}$ مجموعه R و T به ترتیب دو زیر مجموعه R_1 و R_0 و $\hat{T}_1$ و $\hat{T}_0$ تقسیم می‌شود:

$$R_j = \{r \in R \mid Er_{2,i} = j\}$$

$$\hat{T}_j = \{T_i \in T \mid r_i \in R_j\}$$

میانگین دو مجموع $\hat{T}_0$ و $\hat{T}_1$ محاسبه می‌شود، که ترتیب $\bar{T}_0$ و $\bar{T}_1$ نامیده می‌شود. برای تشخیص بیت از $diff = \bar{T}_1 - \bar{T}_0$ استفاده می‌شود. در ادامه نشان داده خواهد شد مقدار $diff$ وابسته به مقدار واقعی β_i و حمله کننده با بررسی آن می‌تواند این بیت کلید را حدس بزند.

برای درنظر گرفتن تاثیر مراحل بر روی همدیگر از مفهمومی به اسم همسایگی استفاده می‌کنیم. برای یک Er، سایر Erها که قبل از آن احتمالا رخ داده‌اند یا بعدا در طی روند الگوریتم احتمال رخ دادن آنها وجود دارد، تشکیل یک همسایگی را می‌دهند. یک همسایگی $N=\{a,b\}$ برای یک Er شامل a عدد Er قبلی و b عدد Er بعدی می‌شود و تاثیر سایر Erها بر روی رخ دادن یا ندادن این Er را در نظر نمی‌گیریم. برای مثال همسایگی $N=\{1,1\}$ برای $Er_{2,i}$ شامل $Er_{1,i}$ (به عنوان یک Er قبلی که ممکن است رخ داده باشد) و $Er_{1,i-1}$ (به عنوان یک Er بعدی که احتمال رخ دادن آن وجود دارد) می‌شود.

اگر همسایگی $N=\{0,0\}$ برای $Er_{2,i}$ در نظر بگیریم. متوسط زمان اجرای مجموعه‌های R_1 و R_0 در دو حالت قابل بررسی هستند، زمانی که فرض $\beta_i=1$ درست است و زمانی که این فرض صادق نباشد. در حالتی که فرض $\beta_i=1$ درست باشد زمان اجرا واقعی اعضای هر یک از مجموعه‌ها به این صورت خواهد بود:

$$\forall r_m \in R_0 \rightarrow T_m = \sum_{j=159}^{i+1}[t_1+\alpha_j t_2+\beta_j(t_1+\gamma_j t_2)] + t_1+\alpha_i t_2+t_1+\sum_{j=i-1}^{0}[t_1+\alpha_j t_2+\beta_j(t_1+\gamma_j t_2)]$$

$$\forall r_m \in R_1 \rightarrow T_m = \sum_{j=159}^{i+1}[t_1+\alpha_j t_2+\beta_j(t_1+\gamma_j t_2)] + t_1+\alpha_i t_2+t_1+t_2+\sum_{j=i-1}^{0}[t_1+\alpha_j t_2+\beta_j(t_1+\gamma_j t_2)]$$

که برای سادگی کار به صورت زیر نشان می‌دهیم:

$$\forall r_m \in R_0 \rightarrow T_m = A_{m,0}+t_1+\alpha_i t_2+t_1+B_{m,0}$$
$$\forall r_m \in R_1 \rightarrow T_m = A_{m,1}+t_1+\alpha_i t_2+t_1+t_2+B_{m,1}$$

متوسط زمان اجرا برای هر مجموعه به صورت زیر محاسبه می‌شود:

$$\bar{T_0}=\frac{\sum A_{m,0}}{k_0}+2t_1+\frac{\sum \alpha_i}{k_0}t_2+\frac{\sum B_{m,0}}{k_0}$$
$$\bar{T_1}=\frac{\sum A_{m,1}}{k_1}+2t_1+\frac{\sum \alpha_i}{k_1}t_2+t_2+\frac{\sum B_{m,1}}{k_1}$$

که در آن k_0 و k_1 به ترتیب اندازه مجموعه R_0 و R_1 هستند. با استفاده از فرض استقلال مراحل خواهیم داشت:

$$\lim_{k\to\infty} diff_1 = \lim_{k\to\infty}(\bar{T_1}-\bar{T_0}) = \lim_{k\to\infty}\left(\frac{\sum A_{m,1}}{k_1}-\frac{\sum A_{m,0}}{k_0}\right)+\lim_{k\to\infty}\left(\frac{\sum \alpha_i}{k_1}t_2-\frac{\sum \alpha_i}{k_0}t_2\right)+\lim_{k\to\infty}\left(\frac{\sum B_{m,1}}{k_1}-\frac{\sum B_{m,0}}{k_0}\right)+t_2=t_2$$

زیرا با این فرض، رخ دادن $Er_{2,i}$ هیچ تاثیری در رخ دادن Erهای قبلی و بعدی ندارد پس میانگین $A_{m,1}$ و $A_{m,0}$ وقتی k به‌اندازه کافی بزرگ باشد مقدار یکسانی خواهد داشت، برای B و α نیز این امر صادق است.

زمانی که فرض $\beta_i=1$ صحیح نباشد، زمان اجرا واقعی اعضای هر یک از مجموعه‌ها به صورت زیر خواهد بود:

$$\forall r_m \in R_0 \rightarrow T_m = \sum_{j=159}^{i+1}[t_1+\alpha_j t_2+\beta_j(t_1+\gamma_j t_2)] + t_1+\alpha_i t_2+\sum_{j=i-1}^{0}[t_1+\alpha_j t_2+\beta_j(t_1+\gamma_j t_2)]$$

$$\forall r_m \in R_1 \rightarrow T_m = \sum_{j=159}^{i+1}[t_1+\alpha_j t_2+\beta_j(t_1+\gamma_j t_2)] + t_1+\alpha_i t_2+\sum_{j=i-1}^{0}[t_1+\alpha_j t_2+\beta_j(t_1+\gamma_j t_2)]$$

با روند مشابهی که $diff_1$ محاسبه شد خواهیم داشت:

$$\lim_{k\to\infty} diff_1 = \lim_{k\to\infty}(\bar{T_1}-\bar{T_0}) = \lim_{k\to\infty}\left(\frac{\sum A_{m,1}}{k_1}-\frac{\sum A_{m,0}}{k_0}\right)+\lim_{k\to\infty}\left(\frac{\sum \alpha_i}{k_1}t_2-\frac{\sum \alpha_i}{k_0}t_2\right)+\lim_{k\to\infty}\left(\frac{\sum B_{m,1}}{k_1}-\frac{\sum B_{m,0}}{k_0}\right)=0$$

بنابراین مقدار $diff$ وابسته به بیت کلید است و تحلیلگر با استفاده از مقدار محاسبه شده $diff$ بیت کلید را حدس می‌زند. اگر مقدار محاسبه شده $diff$ نزدیک $diff_0$ باشد بیت کلید صفر است و اگر این مقدار نزدیک $diff_1$ باشد بیت کلید ۱ حدس زده می‌شود. نکته بسیار مهمی که وجود دارد شرطی است که در این محاسبات وجود داشت. این نتایج با همسایگی $N=\{0,0\}$ بدست آمد.

با محاسبات مشابه، می‌توان نشان داد که هرچه همسایگی بزرگتر شود مقدار $diff_0$ افزایش و مقدار $diff_1$ کاهش می‌یابد و برای همسایگی $N=\{2,2\}$ و بزرگتر از آن، $diff_0$ بزرگتر از $diff_1$ خواهد بود. مقدار $diff_0$ و $diff_1$ به ازای چندین همسایگی مختلف در جدول ۱ نشان داده شده است.

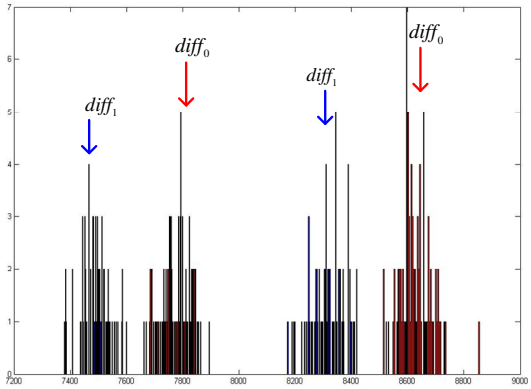
جدول ۱- مقدار $diff_0$ و $diff_1$ برای پنجره‌های مختلف

پنجره	$\{0\}$	$\{1,1\}$	$\{2,2\}$
$diff_0$	0	$\frac{4}{16}t_2$	$\frac{1}{2}t_2$
$diff_1$	t_2	$\frac{7}{16}t_2$	$\frac{1}{3}t_2$

پس برای حدس زدن مقدار بیت کلید، تحلیلگر مقدار $diff$ را بررسی می‌کند اگر مقدار بزرگی باشد بیت صفر حدس زده می‌شود در غیر این صورت این بیت یک خواهد بود. مسئله این است که حمله کننده معیاری برای تشخیص بزرگی یا کوچکی در اختیار ندارد. در ادامه روشی برای رفع این مشکل ارایه می‌شود. فرض می‌کنیم تحلیلگر ماشینی مشابه ماشین تحت حمله در اختیار دارد و می‌تواند به تعداد دلخواهی پیام تولید کرده و الگوریتم را با هر کلید مخفی دلخواه اجرا کند. در ضمن وی توانایی اندازه‌گیری زمان کل اجرای الگوریتم را دارد.

تحلیلگر برای بدست آوردن تخمینی برای این مقادیر از نتایج شبیه‌سازی بر روی ماشین خود استفاده می‌کند. تحلیلگر از مجموعه R که از ماشین تحت جمع‌آوری کرده است استفاده می‌کند و به ازای چند کلید مخفی تصادفی الگوریتم را اجرا می‌کند.

شکل ۴ توزیع $diff_0$ و $diff_1$ را به ازای یک مجموعه پیام و دو مقدار متفاوت کلید مخفی نشان می‌دهد. این شکل وابستگی کامل $diff_0$ و $diff_1$ به کلید مخفی را نشان می‌دهد. بنابراین تحلیلگر بدون داشتن کلید مخفی نمی‌تواند $diff_0$ و $diff_1$ را با شبیه سازی بدست آورد.



شکل۴- توزیع تفاضلها به ازای دو کلید متفاوت

نکته مهم دیگری در این شکل وجود دارد این است که وقتی کلید مخفی ثابت است $diff_0$ و $diff_1$ حول یک مقدار میانگین پراکنده شده‌اند.برای رفع این مشکل، برای تصمیم‌گیری در مورد مقدار بیت به جای استفاده از مقادیر $diff$ از تغییرات آن استفاده می‌کنیم. تحلیلگر تمام بیتهای ۱۵۹ تا $i+1$ را دارد و می‌خواهد بیت i را بیابد. اگر بیت $i+1$ برابر ۰ باشد، دو حالت برای بیت i وجود دارد:

۱- اگر بیت i برابر صفر باشد، مقدار $diff$ محاسبه شده برای آن که با $diff^{\,i}$ نشان می‌دهیم تقریبا برابر $diff$ مرحله قبلی، $diff^{\,i+1}$ است و اختلاف بین آنها، $diff^{\,i+1}-diff^{\,i}$، تقریبا برابر صفر خواهدبود.

۲- اگر بیت i برابر یک باشد، مقدار $diff^{\,i}$ کمتر از $diff^{\,i+1}$ است و $diff^{\,i+1}-diff^{\,i}$ یک مقدار کوچکتر از صفر خواهد بود.

برای حالتی که بیت $i+1$ یک باشد به صورت مشابه خواهیم داشت:

۱- اگر بیت i برابر صفر باشد، $diff^{\,i+1}-diff^{\,i}$ مقدار بزرگتراز صفر است.

۲- اگر بیت i یک باشد، $diff^{\,i+1}-diff^{\,i}$ نزدیک صفر خواهد بود.

برای ادامه کار متغیر تصادفی Δ را به این صورت تعریف می‌کنیم:

$$\Delta = diff^{\,i}-diff^{\,i+1}$$

با استفاده از این متغیر جدید، مرحله تصمیم‌گیری حمله به این صورت تغییر خواهد کرد:

مقدار Δ محاسبه می‌شود، اگر بیت قبلی یک باشد و Δ مقداری "نزدیک به صفر" باشد این بیت نیز یک است. اگر بیت قبلی یک باشد و Δ مقداری "خیلی بزرگتر از صفر" باشد این بیت صفر حدس زده می‌شود. اگر بیت قبلی صفر باشد و Δ "نزدیک به صفر" باشد، این بیت صفر حدس زده می‌شود و اگر بیت قبلی صفر بوده و Δ "خیلی کمتر از صفر" است، این بیت یک است.

در این تصمیم‌گیری نیز مشابه مرحله تصمیم پیش‌گفته، معیارهای تصمیم‌گیری کاملا دقیق نیستند و از معیارهایی مانند "نزدیک به صفر" و "خیلی بزرگتر از صفر" استفاده شده است. این محدودها توسط مرزی از هم جدا می‌شوند. تحلیلگر بایستی دو مرز را بدست آورد، یک مرز برای جدا کردن این دو محدوده در حالتی که بیت قبلی صفر است و مرز دوم برای حالتی که بیت قبلی یک است.

برعکس مشکلی در حالت قبل وجود داشت، در اینجا تحلیلگر می‌تواند از ماشین خود برای اندازه‌گیری زمان اجرا و بدست آوردن این دو مرز استفاده کند. زیرا با توجه به آزمایشهای تجربی وابستگی متغیر تصادفی Δ به بیتهای کلید بسیار کمتر از وابستگی متغیر تصادفی $diff$ است. مرز بین محدوده‌ها به این صورت بدست می‌آید:

۱- تحلیلگر مجموعه کلیدهای مخفی را ایجاد می‌کند.

$$X = \{x_1, x_2, ..., x_n\}$$

۲- به ازای هر کلید مخفی، الگوریتم را برای تمام اعضای R اجرا کرده و زمان اجرا را بدست می‌آورد.

$$T' = \{T_1', T_2', ..., T_n'\}$$

$$T_i' = \{t_1', t_2', t_3', ..., t_k'\}$$

$$t_j' := Time(ALGORITHM(x_i, r_j))$$

۳- با استفاده از مجموعه T_i' و اجرای حمله بر روی این مقادیر مجموعه D به صورت زیر ایجاد می‌شود:

$$D = \{\Delta_{j,159}, \Delta_{j,158}, ..., \Delta_{j,0} \mid 1 \le j \le n\}$$

که در آن $\Delta_{j,i}$، Δ ی است که برای بیت i_ام با استفاده T_j' محاسبه شده است.

۴- مجموعه D به چهار زیر مجموعه تقسیم می‌شود:

$$\Delta^{00} = \{\Delta_{j,w} \mid x_{j,i+1} = 0, x_{j,i} = 0,\ 1 \le j \le n, 159 \le w \le 0\}$$

$$\Delta^{01} = \{\Delta_{j,w} \mid x_{j,i+1} = 0, x_{j,i} = 1,\ 1 \le j \le n, 159 \le w \le 0\}$$

$$\Delta^{10} = \{\Delta_{j,w} \mid x_{j,i+1} = 1, x_{j,i} = 0,\ 1 \le j \le n, 159 \le w \le 0\}$$

$$\Delta^{11} = \{\Delta_{j,w} \mid x_{j,i+1} = 1, x_{j,i} = 1,\ 1 \le j \le n, 159 \le w \le 0\}$$

که در آن $x_{j,i}$ بیت i از کلید j است.

۵- Δ^{00} بیانگر محدوده "نزدیک به صفر" و Δ^{01} بیانگر محدوده "خیلی کمتر از صفر" در حالتی که بیت قبلی صفر باشد است. با استفاده از توزیع آماری Δ^{00} و Δ^{01} مرزی برای جدا کردن این دو محدوده دست می‌آید و به صورت مشابه با استفاده از Δ^{10} و Δ^{11} مرز بین دو مجموعه محاسبه می‌شود.

بعد از اینکه تحلیلگر توانست مرز بین مجموعه‌ها را مشخص کند، با استفاده از تصمیم‌گیری شرح داده شد می‌تواند بیتهای کلید را تشخیص دهد.

توجه به این نکته حائز اهمیت است که تحلیلگر از ماشین خود تنها برای محاسبه تخمینی برای معیارهای حمله استفاده می‌کند و برای اجرای حمله تنها از اطلاعات زمانی که از ماشین تحت حمله جمع‌آوری شده است استفاده می‌شود. این حالت متفاوت با حمله‌هایی مانند حمله مطرح شده توسط Kocher است. در حمله‌ای که Kocher مطرح کرد، تحلیلگر ماشینی مشابه به ماشین تحت حمله در اختیار دارد و دارای این توانایی نیز هست که زمان اجرای الگوریتم را تا هر مرحله دلخواهی بر روی ماشین خود اندازه‌گیری کند. این فرض در اینجا وجود ندارد، تحلیلگر نیازی به اندازه‌گیری زمان تا مرحله دلخواهی را ندارد، بلکه تنها از زمان اجرای کل استفاده می‌کند. این مسئله می‌تواند در محیطهایی مانند smart card یا زمانی که تحلیلگر نتواند به دلخواه خود کد برنامه را دستکاری کند مفید خواهد بود.

۵- نتایج عملی

برای اجرای حمله به صورت عملی ما از سیستم‌عامل DOS که بر روی یک ماشین Athlon XP اجرا می‌شود برای اندازه‌گیری زمان استفاده کردیم. الگوریتم ضرب پیمانه‌ای Blakley ۱۶۰ بیتی با استفاده از توابع اولیه موجود در پیاده‌سازی مرجع RSA که توسط شرکت RSA Data Security Inc. ارایه شده است و به اسم RSAREF2.0 شناخته می‌شود پیاده‌سازی شد. برای اندازه‌گیری زمان از شمارنده ۶۴ بیتی داخلی پردازنده کردیم. تمام شکلها و نتایجی که در اینجا ارائه می‌شود با مجموعه R با اندازه ۱۵۰۰۰۰۰ بدست آمده است.

اولین مرحله برای حمله، بدست آوردن دو مرز جدا کننده محدوده‌ها است. بایستی مرز بین مجموعه Δ^{00} و Δ^{01} و مرز بین مجموعه Δ^{10} و Δ^{11} مشخص

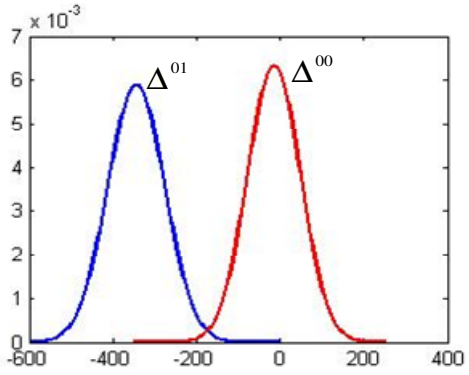
شود. در اینجا از هفت کلید تصادفی مخفی استفاده شده است. با توجه به اینکه هر کلید ۱۶۰ بیتی است و احتمال هر یک از رشته بیتهای 00، 01، 10 و 11 برابر است بنابراین اندازه هریک از مجموعه‌های Δ^{xx} تقریبا ۷×(۱۶۰/۴)=۲۸۰ است. بعد از تولید مجموعه‌های Δ^{00}، Δ^{01}، Δ^{10} و Δ^{11} بایستی توزیع آماری این مجموعه‌ها مورد بررسی قرار گیرد. با فرض نرمال بودن توزیع این مجموعه‌ها، برازش آنها به توزیع نرمال با سطح اطمینان ۹۵ درصد به صورت زیر خواهد بود:

Δ^{00} : μ = -14.7400 σ=63.0916

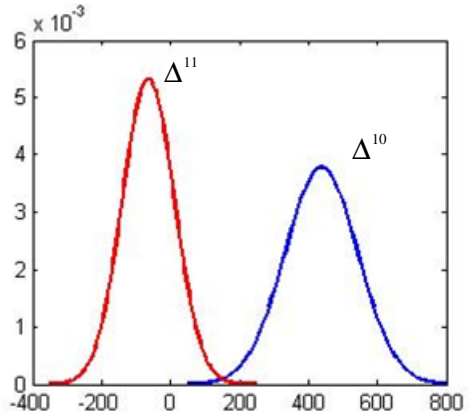
Δ^{01} : μ = -344.196 σ=67.7540

Δ^{10} : μ = 437.6841 σ=105.6146

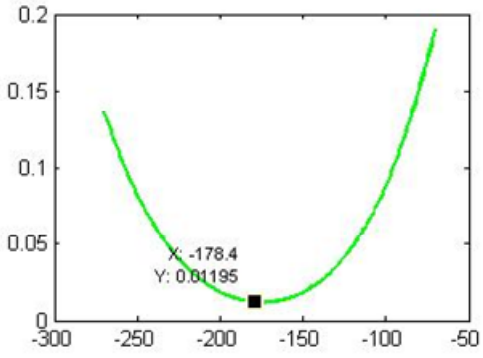
Δ^{11} : μ = -64.2959 σ=74.8556



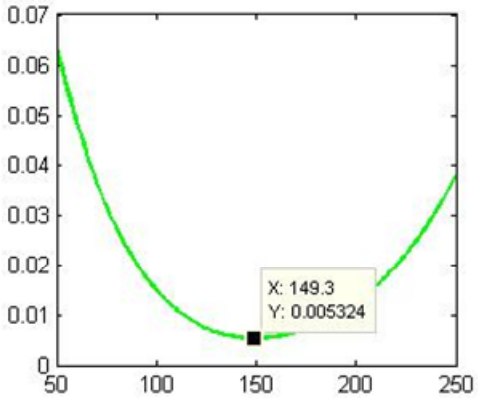
شکل۵- توزیع وقتی بیت قبلی ۰ است



شکل۶- توزیع وقتی بیت قبلی ۱ است



شکل۷- نمودار خطا وقتی بیت قبلی ۰ است



شکل۸- نمودارخطا وقتی بیت قبلی ۱ است

شکل ۵ و ۶ توزیع این مجموعه ها را نشان می‌دهد. مرزی که بین مجموعه‌ها انتخاب می‌شود، بایستی به گونه‌ای باشد که احتمال خطا در تصمیم‌گیری را کاهش دهد. خطا زمانی رخ می‌دهد که برای مثال Δ محاسبه شده در مرحله تصمیم‌گیری حمله، در حقیقت متعلق به مجموعه Δ^{01} است ولی به اشتباه فرض می‌شود که متعلق به Δ^{00} است. با توجه به مقدار بیت قبلی خطا به صورت زیر تعریف می‌شود:

$$E(z)=\begin{cases} F(\Delta^{00},z)+(1-F(\Delta^{01},z)) & if\ x_i=0 \\ (1-F(\Delta^{11},z))+F(\Delta^{10},z) & if\ x_i=1 \end{cases}$$

که در آن $F(\Delta,z)$، مساحت زیر منحنی توزیع احتمال مجموعه Δ از $-\infty$ تا نقطه z است. با فرض توزیع نرمال برای مجموعه‌های Δ^{00}، Δ^{01}، Δ^{10} و Δ^{11} و داشتن پارامترهای توزیع احتمال، مقدار کمینه برای خطا در دو حالت بدست می‌آید. شکل ۷ و ۸ نمودار خطا را برای هر دو حالت نشان می‌دهند.

نقاطی که بر روی شکل مشخص شده است، نقاط کمینه این نمودارها (که به عنوان مرز انتخاب می‌شوند) و احتمال خطای متناظر با آنها است. با توجه به شکل ۷ مرز بین Δ^{00} و Δ^{01} برابر است با 174.4- و احتمال خطای آن نیز برابر است با 0.0120 و بر اساس شکل ۸ مرز مجموعه‌های Δ^{10} و Δ^{11}، 149.3 انتخاب می‌شود که در این حالت احتمال خطای برابر 0.0053 را در تصمیم‌گیری خواهیم داشت.

با مشخص شدن مرزها می‌توان حمله زمانی را انجام داد، در مرحله تصمیم‌گیری حمله، اگر بیت قبلی صفر باشد و Δ محاسبه شده کمتر از 174.4- باشد، این بیت ۱ حدس زده می‌شود در غیر این صورت مقدار آن صفر فرض می‌شود. اگر بیت قبلی ۱ باشد مقدار Δ با 149.3 مقایسه می‌شود، اگر از این مقدار بزرگتر باشد، بیت جاری صفر است و در صورت کوچکتر بودن این بیت ۱ فرض می‌شود. نتایج این طرح حمله برای چهار مجموعه داده مختلف در جدول ۲ ارایه شده است. شکل ۹ نمودار Δ را نشان می‌دهد. مقدار Δ برای هر بیت و بیت متناظر با آن نشان داده شده است، همچنین دو مرزی که انتخاب شده است نیز مشخص است. این نمودار نشان می‌دهد که مقدار بیت را می‌توان به درستی بر اساس مقدار بیت قبلی و از روی مقایسه مقدار Δ با مرزها مشخص کرد.

برای نمونه اگر فرض کنیم که می‌دانیم مقدار ۱۰۶ برابر ۱ است، بیت ۱۰۵ صفر خواهد بود زیرا مقدار Δ از مرز بزرگتر است، مقدار Δ در بیت ۱۰۴ کمتر از مرز Δ^{00} با Δ^{01} است پس این بیت ۱ خواهد بود. فرضی که براساس آن مقدار مرزها بدست آمد، نرمال بودن توزیع مجموعه‌های Δ^{xx} است. برای بررسی نرمال بودن توزیع این مجموعه‌ها از تست نرمال Lilliefors با سطح معنی‌دار پنج درصد استفاده کردیم. تمام مجموعه‌ها این تست را با موفقیت پاس کردند. اما در

برخی از داده‌ها این تست‌ها با موفقیت انجام نمی‌شود و داده‌ها توزیع نرمالی از خود نشان نمی‌دهند. در این‌گونه موارد احتمال خطا بیشتر از آن مقداری است که از نمودار خطا بدست می‌آید و در برخی از موارد در حین انجام حمله دچار خطا می‌شویم. برای بهبود تعیین مرزها بایستی اندازه مجموعه R و X بزرگتر شود. بزرگتر کردن این مجموعه به معنای طولانی‌تر شدن زمان شبیه‌سازی‌ها است. بنابراین تحلیلگر با صرف زمان بیشتر برای تعیین مرزهای تصمیم‌گیری احتمال خطا در حین حمله را کاهش می‌دهد. البته می‌توان بدون برازش داده‌ها و استفاده از تحلیل آماری با روشهای هوشمند نیز مرز بین دو مجموعه را مشخص کرد.

جدول ۲- نتایج طرح حمله برای چهار مجموعه داده

پارامترها	داده ۱	داده ۲	داده ۳	داده ۴
$\min(\Delta^{00})$	-89	-149	-73	-81
$\max(\Delta^{01})$	-143	-154	-135	-111
$\max(\Delta^{11})$	44	118	51	18
$\min(\Delta^{10})$	125	179	177	161
تعداد بیت خطا، بیت قبلی صفر	0	1	0	0
تعداد بیت خطا، بیت قبلی یک	1	0	0	0

۶- تشخیص و تصحیح خطا

این حمله مانند اکثر حملات side-channel، بیتهای کلید را یک به یک پیدا می‌کند و حدسی که برای یک بیت زده می‌شود، بستگی به درستی حدسهای قبلی دارد. اگر یکی از بیتهای قبلی خطا باشد، دسته‌بندی پیامها بر اساس یک کلید غلط انجام می‌شود و دسته‌بندی بی‌معنی خواهد بود و هیچ تضمینی برای درستی حدسی که در مراحل بعد از حدس غلط انجام می‌گیرد وجود ندارد.

تشخیص و تصحیح خطا در این نوع حملات با ارزش است. طرح پیشنهادی حمله دارای قابلیت تشخیص و تصحیح خطا است. فرض کنید در فرآیند حمله بیت i اشتباه حدس زده شود، در این صورت در حدس بیت بعدی دسته‌بندی مجموعه R بر اساس یک کلید غلط انجام می‌شود. اما بازهم به علت همبستگی بین مراحل این دسته‌بندی همپوشانی زیادی با دسته‌بندی دارد که اگر بیت کلید درست حدس زده شده بود انجام می‌شد. هر چه بیشتر پیش می‌رویم تاثیر مراحل قبلی کمتر می‌شود و بعد از چندین مرحله مقدار $diff$ به $\frac{diff_0 + diff_1}{2}$ میل می‌کند و درنتیجه مقدار Δ به صفر میل خواهد کرد. شکل ۱۰ نمونه‌ای از حدس غلط است. این شکل مقدار $diff$ در هر مرحله و مقدار واقعی بیت متناظر با آن را نشان می‌دهد. بیت ۳۸ که با فلش نشان داده شده است، بیتی است که به اشتباه حدس زده شده، بعد از چند بیت، همانطور که در کادر نشان داده شده است، مقدار $diff$ تغییرات کمی دارد.

برای تشخیص خطا مقدار $diff$ و Δ را در حین انجام حمله زیر نظر قرار می‌دهیم، اگر برای چندین بیت متوالی (که آن را اندازه پنجره می‌نامیم) Δ تقریبا صفر باشد و $diff$ نیز حوالی $\frac{diff_0 + diff_1}{2}$ باشد، خطا رخ داده است، بنابراین حمله قطع می‌شود. نتایج تجربی ما نشان می‌دهد که اندازه پنجره ۱۰ تایی تقریبا تمام خطاها را تشخیص می‌دهد.

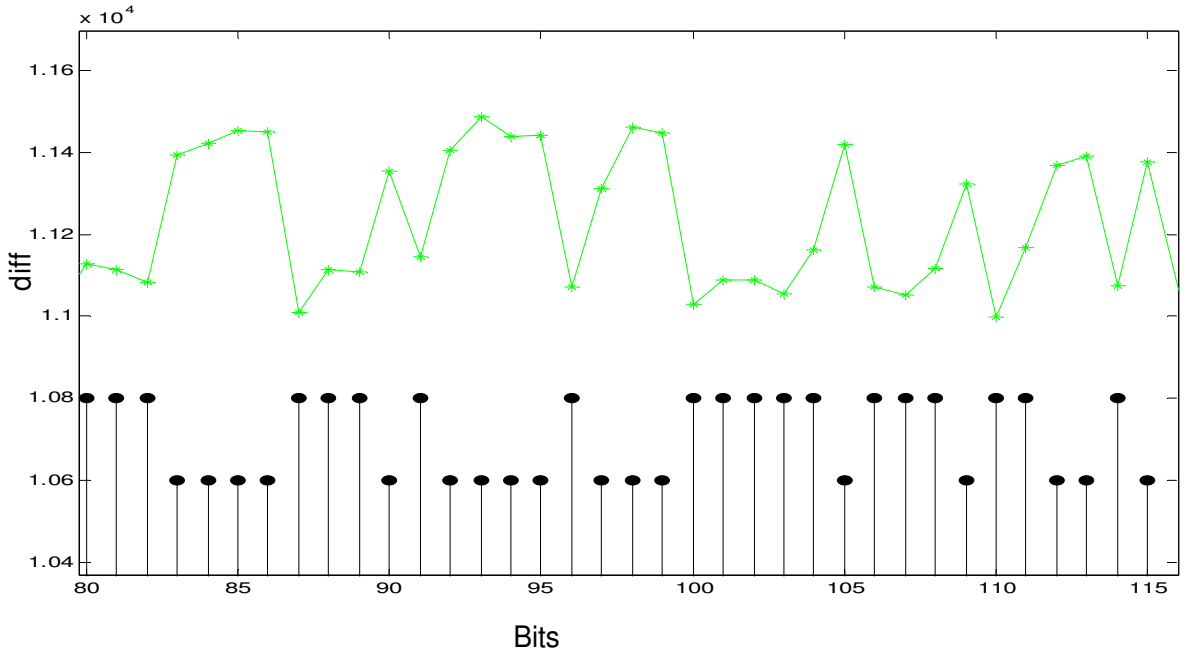
وقتی خطا تشخیص داده شد. برای تصحیح خطا بایستی بیتی که غلط حدس زده شده است را پیدا کرد. بازهم نتایج تجربی نشان می‌دهد که بیتی که خطا در آن رخ داده است حداکثر ۵ بیت قبل از شروع پنجره قرار دارد. این پنج بیت را براساس مقدار نزدیکی Δ آن بیت با مقدار مرز مرتب می‌کنیم. بیتی که مقدار Δ آن به مرز نزدیکتر باشد، احتمال رخ دادن خطا در آن بیشتر است. بنابراین این بیت به عنوان کاندید خطا انتخاب می‌شود، مقدار آن برعکس می‌شود (اگر صفر حدس زده شده تبدیل به ۱ می‌شود و برعکس و حمله دوباره از بیت بعدی شروع می‌شود. اگر دوباره خطا رخ دهد و پنجره خطا با پنجره قبلی همپوشانی زیادی داشته باشد (بیشتر از ۵۰ درصد) کاندیدی که انتخاب شده بود، درست نبوده است، بنابراین کاندید بعدی انتخاب می‌شود و این فرآیند تکرار می‌شود

۷- جمع‌بندی

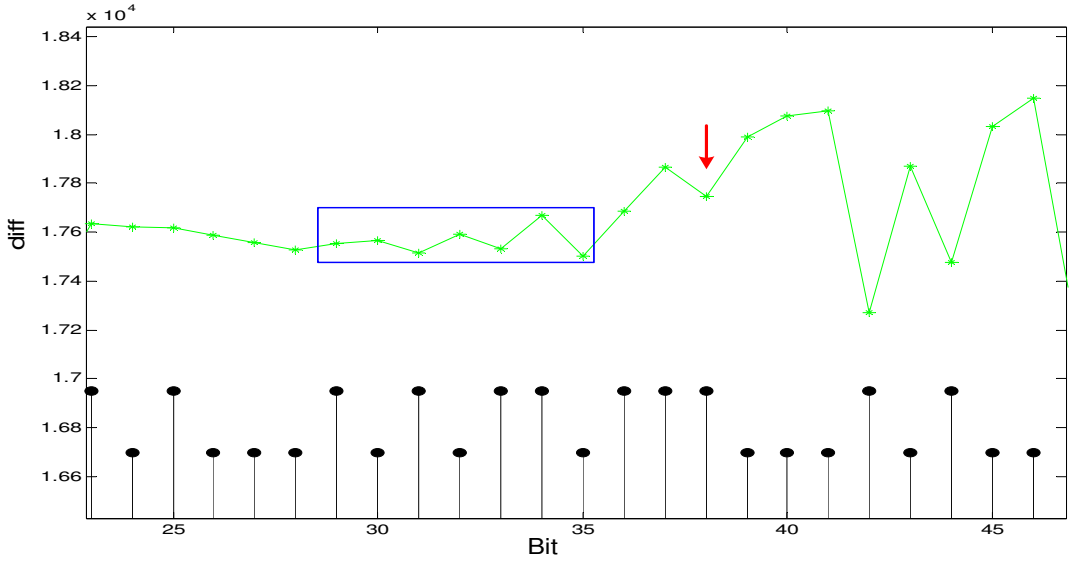
در این مقاله تحلیل زمانی الگوریتم ضرب Blakley ارایه شد. در برخی از کاربردهای رمزنگاری در فرایند انجام الگوریتم، یکی از اجرای مخفی سیستم با یک مقدار عمومی ضرب می‌شود و نتیجه آن دوباره مورد پردازش قرار می‌گیرد. در این مقاله نشان داده شده است که اگر این ضرب با استفاده از الگوریتم Blakley انجام گیرد. تحلیلگر می‌تواند بدون اطلاع از نتیجه عمل ضرب و تنها با دانستن یکی از عملوندهای ضرب و اندازه‌گیری زمان اجرای الگوریتم عملوند دیگر ضرب پیمانه‌ای را بیابد. تغییرات زمان اجرای الگوریتم ضرب Blakley نسبت به الگوریتمهایی مانند توان‌رسانی پیمانه‌ای بسیار کمتر است، این مسئله باعث می‌شود تا نسبت سیگنال به نویز تا حد زیادی کاهش یابد و به تبع آن تعداد نمونه‌های زمانی لازم برای تحلیل زمانی افزایش یابد. یکی از مشکلات اصلی در تحلیل این الگوریتم، همبستگی زیاد بین زمان اجرای مراحل مختلف الگوریتم است. با توجه به اینکه در حملات زمانی بیتهای کلید به ترتیب و بیت به بیت حدس زده می‌شود، استقلال زمان اجرای مراحل تاثیر چشم‌گیری بر روی حملات زمانی دارد. در الگوریتم Blakley فرض استقلال زمان اجرای مراحل صادق نیست، همین مسئله باعث پیچدگی تحلیل می‌گردد. برای مثال آزمایشهای تجربی ما نشان داد که طرح حمله‌ای که Kocher مطرح کرده بود اصلا قابل اعمال در مورد این الگوریتم نیست. این همبستگی به صورت ریاضی مدل شده و تاثیر آن بر روی حمله زمانی بررسی شده است. در طرحی که برای حمله مطرح شد، پیام‌ها بر اساس یک ویژگی از سیستم (اجرا شدن یا اجرا نشدن تفریق اضافی در صورت یک بودن بیت کلید) به دو زیر مجموعه تقسیم‌بندی می‌شوند. در این مقاله نشان داده شد که تفاضل بین متوسط زمان اجرای این دو مجموعه رابطه مستقیم با بیت کلید دارد. اگر این مقدار بزرگ باشد به معنی صفر بودن بیت و اگر کوچک باشد به معنی یک بودن بیت است. مشکل اصلی مشخص کردن مرز بین بزرگ یا کوچک است. به خاطر همبستگی بین مراحل مقدار تفاضل به سایر بیتهای کلید مخفی نیز وابسته است و مشخص کردن آن کار ساده‌ای نیست.

اما مقدار تغییرات تفاضل مستقل از کلید مخفی عمل می‌کند، با فرض اینکه تحلیلگر ماشین مشابه ماشین تحت حمله دارد، می‌تواند مقدار تغییرات را مشخص کرده و از آن برای حمله استفاده کند. در این مقاله الگوریتمی برای یافتن حدود تغییرات ارایه شده و نتایج عملی این الگوریتم و نتایج حمله‌ای بر اساس این الگوریتم بدست آمده است نیز ارایه شده است. علاوه بر طرح حمله، روشی برای تشخیص و تصحیح خطا در حین فرایند اجرای تحلیل ارایه شده است. روش تشخیص ارایه شده با بررسی مقدار تفاضل و تغییرات سعی در تشخیص خطا دارد. نتایج عملی نشان می‌دهد که این روش تمام خطاها را تشخیص می‌دهد. بعد از تشخیص خطا با روشی که ارایه شده محل خطا پیدا شده و تصحیح می‌شود.

.



شکل ۹- نمودار Δ



شکل ۱۰- نمودار *diff* با رخ دادن خطا

مراجع

[1] J. A. Muir, "Techniques of Side Channel Cryptanalysis," *Master Thesis, University of California*, 2001.

[2] P. C. Kocher, "Timing Attacks on Implementations of Diffie--Hellman, RSA, DSS, and Other Systems," *Advances in Cryptology - CRYPTO '96*, pp. 104-113, Springer-Verlag, LNCS Nr. 1109, 1996.

[3] J. F. Dhem, et al, "A Practical Implementation of the Timing Attack," *3rd Working Conference on Smart Card Research and Advanced Applications - CARDIS*, Springer-Verlag, LNCS Nr. 1820, 1998.

[4] W. Schindler, "A Timing Attack against RSA with the Chinese Remainder Theorem," *Cryptographic Hardware and Embedded Systems - CHES 2000*, pp. 109-124, Springer-Verlag, LNCS Nr. 1965, 2000.

[5] C. D. Walter, S. Thompson, "Distinguishing Exponent Digits by Observing Modular Subtractions," *CT-RSA 2001*, LNCS 2020, pp. 192-207, 2001.

[6] H.Sato, D. Schepers, and T. Takagi, "Exact Analysis of Montgomery Multiplication", *INDOCRYPT 2004*, LNCS 3348, pp. 290–304, 2004.

[7] A. Hevia, M. Kiwi, "Strength of Two Data Encryption Standard Implementations Under Timing Attacks", *LATIN'98*, LNCS 1380, pp. 192-205, 1998.

[8] H. Handschuh, H. M. Heys, "A Timing Attack on RC5", *SAC'98*, LNCS 1556, pp. 306-318, 1999.

[9] F. Koeune, J. J. Quisquater, "A Timing Attack against Rijndael", Technical Report CG-1999/1, June 1999.

[10] D. Brumley, D. Boneh, "Remote Timing Attacks are Practical", *12th Usenix Security Symposium*, pp. 1-14, 2003.

[11] Alfred Menezes, Paul Oorschot, and Scott Vanstone, *Handbook of Applied Cryptography*, CRC Press, October 1996.

[12] Cetin Kaya Koc, "High-Speed RSA Implementation", *RSA Laboratories, RSA Data Security Inc*. November 1994.

بهادر بخشی سراسکانرود در حال حاضر دانشجوی دکتری شبکه‌های کامپیوتری در دانشگاه صنعتی امیرکبیر است. او در سال ۱۳۸۲، درجه کارشناسی رشته مهندسی کامپیوتر گرایش سخت‌افزار را از دانشگاه صنعتی شریف کسب کرده و در همان سال مشغول به تحصیل در مقطع کارشناسی ارشد معماری سیستم‌های کامپیوتری در دانشگاه صنعتی امیرکبیر شد. موضوع تحقیقاتی وی در این دوره حملات کانال جانبی و مشخصا حمله زمانی بود. وی در سال ۱۳۸۴ درجه کارشناسی ارشد را کسب کرده دوره دکتری خود را شروع کرد. زمینه‌های تحقیقاتی وی شبکه‌های بی‌سیم، امنیت در سیستم‌های کامپیوتری و رمزنگاری است.

آدرس پست الکترونیکی ایشان عبارت است از:

bakhshis@ce.aut.ac.ir

دکتر بابک صادقیان مدرک کارشناسی ارشد خود را در سال ۱۳۶۷ از دانشگاه صنعتی امیرکبیر در رشته مهندسی برق (الکترونیک) دریافت کرد و مدرک دکتری خود را در سال ۱۳۷۲ از University College, University of New South Wales کشور استرالیا در رشته علوم کامپیوتر دریافت کرد. زمینه های پژوهشی مورد علاقه وی عبارتند از معماشناسی، بخصوص طراحی و تحلیل الگوریتم های رمز قطعه ای، و امنیت شبکه، بخصوص سیستمهای تشخیص نفوذ، می باشد. او در حال حاضر دانشیار دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر است.

آدرس پست الکترونیکی ایشان عبارت است از:

basadegh@ce.aut.ac.ir