

معرفی حمله بده‌بستان زمان-حافظه (TMTO) بر

توابع چکیده‌ساز

زهرا ذوالفقاری* و منصور باقری

دانشکده مهندسی برق، دانشگاه تربیت دبیر شهید رجایی، تهران، ایران

z.zolfaghari71@gmail.com

nbagheri@srttu.com

چکیده

در این مقاله به معرفی حمله TMTO و نحوه پیدا کردن نزدیک-برخوردها در یک تابع چکیده‌ساز پرداخته شده است. با در نظر گرفتن محاسبات چکیده‌ساز، محاسبه یک حد پایین برای پیچیدگی الگوریتم‌های نزدیک-برخورد و ساخت یک الگوریتم مطابق با آن آسان است؛ با این حال، این الگوریتم نیاز به مقدار زیادی حافظه دارد و موجب استفاده از $2^{\frac{n}{2}}$ حافظه می‌شود. به تازگی، چند الگوریتم بدون نیاز به این مقدار حافظه ارائه شده‌اند. این الگوریتم‌ها نیاز به مقدار بیشتری از محاسبات چکیده‌ساز دارند؛ اما این حمله در واقعیت عملی‌تر است. این دسته از الگوریتم‌ها را به دو دسته اصلی می‌توان تقسیم کرد: گروهی مبتنی بر کوتاه‌سازی و گروهی دیگر مبتنی بر کدهای پوششی هستند. در این کار، بده‌بستان زمان-حافظه برای الگوریتم‌های مبتنی بر کوتاه‌سازی در نظر گرفته شد. برای پیاده‌سازی عملی، می‌توان فرض کرد مقداری از حافظه موجود است و نشان داد که با استفاده از این حافظه قابل توجه پیچیدگی را می‌توان کاهش داد؛ در مرحله بعد، با استفاده از برخوردهای متعدد بر اساس جدول هلمن، بهبود شناخته‌شده‌ترین بده‌بستان زمان حافظه، برای K درخت ارائه شد. در نتیجه، منحنی بده‌بستان جدید $T^2 \cdot M^{1/gk-1} = k \cdot N$ به دست آورده شد، با در نظر گرفتن $k=4$ منحنی بده‌بستان به شکل $T^2 \cdot M = 4 \cdot N$ خواهد بود. در این مقاله، ابتدا روش‌های TMTO و سپس نحوه پیدا کردن نزدیک-برخورد با استفاده از TMTO شرح داده می‌شود.

واژگان کلیدی: تابع چکیده‌ساز، نزدیک-برخورد، بده‌بستان زمان-حافظه

۱- مقدمه

حمله پیش‌تصویر دوم^۳: زمانی که h و x داده شده

است، بتوان $x' \neq x$ پیدا کرد که $h(x)=h(x')$.

یکی از مسائل متداول در رمزنگاری کلید خصوصی

مسئله پیدا کردن برخورد است. اغلب نه‌تنها در پیدا کردن

برخورد برای توابع چکیده‌ساز بلکه در مسائل دیگر رمزنگاری

نیز مطرح است. طی چندین دهه، پیدا کردن برخورد به‌طور

گسترده مورد مطالعه قرار گرفته است. علاوه بر این مسئله و

همراه با پیدا کردن چندین برخورد متعدد، مسئله رایج بعدی

تولد عمومی تعمیم یافته (GBP) است.

با توجه به مسئله تولد، یک حمله برخورد عمومی

دارای پیچیدگی $2^{\frac{n}{2}}$ است؛ در حالی که حمله‌های پیش‌تصویر

یا پیش‌تصویر دوم دارای پیچیدگی 2^n هستند؛ که

نیازمندی‌های امنیتی یک تابع چکیده‌ساز n بیتی را تعریف

توابع چکیده‌ساز، از نخستین رمزنگاری‌های بنیادین مورد

استفاده در بسیاری از ساختارها و پروتکل‌ها هستند. یک تابع

چکیده‌ساز یک رشته بیت به‌طول دلخواه را به‌عنوان ورودی

می‌گیرد و یک رشته بیت با طول ثابت n را در خروجی

تولید می‌کند:

$$h: \{0,1\}^* \rightarrow \{0,1\}^n, \quad (1)$$

هنگامی که در زمینه رمزنگاری استفاده می‌شود، یک

تابع چکیده‌ساز باید در برابر سه حمله اصلی مقاوم باشد:

حمله برخورد^۱: زمانی که h داده شده است، بتوان

$$x' \neq x \text{ پیدا کرد که } h(x)=h(x')$$

حمله پیش‌تصویر^۲: زمانی که h و y داده شده است،

$$\text{بتوان } X \text{ پیدا کرد که } h(x)=y$$

³ Second-preimage attack

¹ Collision attack

² Preimage attack

* نویسنده عهده‌دار مکاتبات

رمزنگاری بسیار مهم تلقی می‌شود؛ اما بیش از یک دهه پس از انتشار آن بهبود قابل توجهی در الگوریتم K-درخت و در دیگر الگوریتم‌های اختصاصی مشاهده نشده است؛ با این حال، پیشرفت‌های جزئی و اصلاحاتی منتشر شده. که یکی از مهم‌ترین آن‌ها الگوریتم K-درخت توسعه‌یافته است که توسط میندر و سینکلر [6] پیشنهاد شده است. این الگوریتم راه حلی را برای GBP زمانی که فهرست‌ها دارای اندازه‌های کوچک‌تر هستند، ارائه می‌کند.

از نمادهای زیر در این مقاله استفاده شده است:

N: اندازه خروجی تابع چکیده‌ساز

T: اندازه خروجی برش‌یافته

W: بیشینه فاصله برای نزدیک-برخوردها

M: اندازه حافظه

$B_w(n)$: اندازه یک توپ همینگ به شعاع w

۲- مروری بر مطالعات مرتبط

در اینجا ابتدا به بحث در مورد حمله TMTO و سپس ترفندهای پیداکردن برخورد کامل پرداخته می‌شود. در ادامه نیز ترفندهای پیداکردن نزدیک-برخورد شرح داده می‌شوند.

۱-۲- الگوریتم‌های جستجوی جامع و TMTO

فرض کنید، مهاجم قصد رمز گشایی کردن پیام رمزگذاری شده m' دارد. هدف محاسبه m برای $f(m)=m'$ است که در آن f می‌تواند یک تابع چکیده‌ساز، رمز قالبی^۲ و یا یک رمز جریایی^۳ باشد. در یک تابع چکیده‌ساز، m نشان‌دهنده متن ورودی است. یک کار مهاجم که همیشه می‌تواند انجام دهد انجام جستجوی جامع است: همه m های ممکن را امتحان کند و ببیند که آیا f درخواستی در m' نتیجه می‌دهد یا نه. رمزشکنی با استفاده از جستجوی جامع به زمان زیادی می‌تواند نیاز داشته باشد. برای یک سامانه رمزنگاری با کلید n بیتی، به‌طور معمول $N=2^n$ کلید ممکن وجود دارد. هنگامی که N به اندازه کافی بزرگ است، امتحان تمام احتمال‌ها غیرممکن می‌شود. با هوشمندسازی تا حد زیادی زمان صرف‌شده برای شکستن در حمله‌های مکرر را می‌توان کاهش داد. به‌عنوان مثال، نتایج رمزگذاری تمام کلیدهای ممکن N را از پیش می‌توان محاسبه و آن‌ها را در یک جدول به‌صورت جفت $\langle m, f(m) \rangle$ ذخیره کرد. این مرحله، برون‌خط حمله نامیده می‌شود. زمانی که یک نفر بخواهد یک پیام

² Block cipher

³ Stream cipher

می‌کنند. به‌طور کلی، انتظار می‌رود که یک تابع چکیده‌ساز مانند یک تابع تصادفی رفتار کند. این شرط به‌طور واقعی نمی‌تواند به رسمیت شناخته‌شود؛ اما انتظار می‌رود هر خاصیتی که در تابع چکیده‌ساز موردنظر می‌توان نشان داد، در یک تابع تصادفی نیز باشد؛ به‌ویژه، انتظار می‌رود که پیداکردن دو پیام و در نتیجه چکیده آن‌ها با یک تفاوت کوچک سخت باشد. این ویژگی نزدیک-برخورد نامیده می‌شود و چندین حمله در این مورد پیشنهاد شده است [1],[2],[3].

انتخاب یک کران پایین برای پیچیدگی حمله‌های نزدیک-برخورد تا حدودی آسان است (دست‌کم به $2^{n/2}/\sqrt{B_w(n)}$ ارزیابی تابع چکیده‌ساز نیاز دارد). با این حال تنها راه شناخته‌شده برای رسیدن به این حد پایین نیاز به مقدار زیادی حافظه و بیش از $2^{n/2}$ حافظه در دسترس دارد. برای حل این مشکل، لمبرگر^۱ و همکاران یک روش با حافظه کمتر مبتنی بر کدهای پوششی [4]، با یک پیچیدگی بین $2^{n/2}$ و $2^{n/2}/\sqrt{B_w/2(n)}$ پیشنهاد کردند. در این کار، مسئله پیداکردن نزدیک - برخورد با الگوریتمی که در عمل به‌طور مؤثر می‌تواند پیاده‌سازی شود، پیشنهاد شد. با توجه به ماشین مورد استفاده برای اجرای این نوع محاسبات بزرگ (خوشه‌ها، GPU ها و یا سخت‌افزار اختصاص داده‌شده)، هدف الزاماً به‌دست‌آوردن یک الگوریتم با حافظه کم‌تر نیست؛ هدف تنها رسیدن به یک الگوریتم با مقدار عملی حافظه و مقدار عملی دسترسی به حافظه است. نتایج نشان می‌دهد که می‌توان به پیچیدگی کم‌تر از پیچیدگی الگوریتم حافظه-کم مبتنی بر کدهای پوششی دست یافت.

GBP به‌صورت زیر تعریف می‌شود: با فرض K فهرست از عناصر تصادفی، یک عنصر را در هر یک از فهرست‌ها انتخاب کنید، به‌طوری که تمام عناصر انتخاب‌شده به یک مقدار از پیش تعریف شده خلاصه شوند. واگنر [5] برای نخستین‌بار به بررسی GBP برای تمام مقادیر K به‌عنوان یک مسئله مستقل پرداخته است. ایشان یک الگوریتم برای حل GBP برای تمام مقادیر K پیشنهاد کرده است و کاربرد آن در طیف گسترده‌ای از برنامه‌های کاربردی اعم از امضای کور، برای چکیده‌سازی تدریجی، بررسی تساوی وزن کم و تحلیل رمز توابع مختلف چکیده‌ساز را نشان می‌دهد. اگرچه GBP برای بسیاری از مشکلات در

¹ Lamberger

$M=2^m$ مقدار حافظه در دسترس است. هنگامی که پیش‌محاسبه با ارزیابی MX از f انجام شود، هزینه تولید برخورد جدید برای f ، به ازای هر برخورد است.

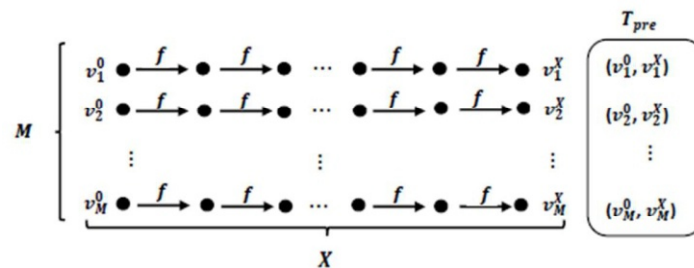
این روش به شرح زیر کار می‌کند:

انتخاب M مقدار متمایز $v_i^0 \in \{0,1\}^n$ که برای $i=1,2,\dots,M$ که برای هر یک از آن‌ها زنجیره‌ای با طول X با تابع هدف f محاسبه می‌شود، یعنی محاسبه $v_i^j \leftarrow f(v_i^{j-1})$ برای $i=1,\dots,M, j=1,\dots,X$ و ذخیره تنها مقدار نخست و آخر هر زنجیره یعنی (v_i^0, v_i^X) در یک جدول پیش‌محاسبه T_{pre} ساختار T_{pre} در شکل (۱) نشان داده شده است.

الگوریتم K-درخت، بر روی تولید برخوردهای متعدد متکی است. به‌عنوان مثال، در سطح نخست از چهار درخت، $2^{\frac{n}{3}}$ برخورد جفت‌ها بر روی $\frac{n}{3}$ بیت تولید می‌شود. تولید این جفت‌ها زمانی که مقدار حافظه در دسترس $2^{\frac{n}{3}}$ باشد، بی‌اهمیت است. با این حال، هنگامی که حافظه به $2^m, m < \frac{n}{3}$ کاهش یابد، پیدا کردن برخورد ناچیز، عملی نیست. الگوریتم K-درخت نیاز به برخوردهای متعدد دارد و بر اساس جدول هلمن راه‌حل‌ها را ارائه می‌کند.

واقعیت ۱ (جدول هلمن) اگر $f: \{0,1\}^* \rightarrow \{0,1\}^n$

یک ورودی دلخواه و تابع خروجی n بیتی باشد، $N=2^n$ و



(شکل-۱): جدول هلمن به T_{pre} هنگامی که حافظه در دسترس M باشد [9].

معمولی و $2^{\frac{2n}{3}}$ مقدار پیدا کردند. هنگامی که آن‌ها $2^{\frac{n}{3}}$ برخورد متعدد معمولی تولید کردند، جدول هلمن نقش مهمی برای حفظ M حافظه به جای MX داشت. علاوه‌براین، از جدول هلمن برای تولید برخوردهای متعدد برای نخستین سطح K-درخت، اما تنها در L بیت خاص (که در آن $n < L$) استفاده کردند.

۱-۳- بهبود بده‌بستان زمان-حافظه برای

مسئله K-فهرست

در این بخش، بده‌بستان زمان-حافظه را برای الگوریتم K-درخت ($K=2D$)، تعمیم‌داده‌شده معرفی می‌شود. به‌طور کلی، تولید برخورد در سطح ۱ الگوریتم K-درخت را با یک تولید براساس جدول هلمن جایگزین کردند. درنهایت بیت‌هایی را که حاصل جمع آن‌ها بیت‌های ذخیره شده ثابت صفر است، فراخوانی می‌کنند.

الگوریتم K-درخت معمولی در ابتدا از 2^d فهرست حاوی عناصر $M = 2^m$ شروع می‌شود. در سطح یک، 2^{d-1} فهرست حاوی M عنصر با m بیت ذخیره‌شده تولید می‌شوند. در سطح i برای فهرست‌های $i = 2, 3, \dots, d$ - 2^{d-i} حاوی M عنصر با im بیت ذخیره‌شده تولید می‌شوند. در آخرین سطح D دو فهرست حاوی M عنصر با

توجه داشته باشید، حتی اگر مقادیر MX در تمام زنجیره‌ها وجود داشته باشد، فقط مقادیر $2M$ در T_{pre} ذخیره می‌شوند. هنگامی که T_{pre} ساخته شود، برای تولید یک برخورد، با یک نقطه تصادفی شروع کرده و یک زنجیره‌ای به طول $\frac{N}{MX}$ ساخته می‌شود. همان‌طور که N مقدار ممکن وجود دارد و MX مقدار در T_{pre} هستند، یک نقطه از زنجیره‌ای جدید با یک نقطه از زنجیره ایجادشده در ساختار جدول برخورد خواهد داشت. این تطابق را با گسترش بیشتر زنجیره‌های جدید تا بیشینه X مرتبه می‌توان شناسایی کرد، تا درنهایت به یکی از v_i^X ذخیره شده در T_{pre} برسد؛ سپس مقادیر دقیق برخورد را با محاسبه مجدد زنجیره‌ها از v_i^0 و مقدار شروع زنجیره جدید می‌توان شناسایی کرد. بدیهی است از T_{pre} دوباره برای پیدا کردن نه تنها یک برخورد، بلکه برخوردهای متعدد می‌توان استفاده کرد.

ژوا^۱ و لوکس^۲ از این روش برای پیدا کردن سه برخورد استفاده کردند. آن‌ها $M = X = 2^{\frac{n}{3}}$ را برای تولید $2^{\frac{n}{3}}$ برخورد معمولی با زمان $T = 2^{\frac{2n}{3}}$ و حافظه $M = 2^{\frac{n}{3}}$ تنظیم و سپس، برخورد دیگری را بین $2^{\frac{n}{3}}$ برخورد

¹ Joux

² Lucks

هزینه $\frac{2^l}{MX}$ در هر برخورد (که معادل $\frac{N}{M^{d+l}}$ است) می‌توان پیدا کرد. در سطح یک، در کل $(2^{d+l} \cdot M)$ برخورد l بیتی تولید و آنها را در 2^{d-l} فهرست هر کدام با M عنصر ذخیره کردند. در کل هزینه برای تولید برخوردهای نسبی و سپس پیچیدگی سطح یک $2^{d-l} \cdot \frac{N}{M^{dX}}$ است.

۲-۳- ارزیابی پیچیدگی

پیچیدگی برای تولید T_{pre} ، MX زمان و M حافظه است. همان‌طور که در بالا ذکر شد، سطح یک نیاز به $2^{d-l} \cdot \frac{N}{M^{dX}}$ زمان و $2^{d-l} \cdot M$ حافظه دارد. زمان و حافظه پیچیدگی‌های سطوح باقی‌مانده دو به دو D همه M هستند، در نتیجه در مقایسه با تولید T_{pre} ناچیز هستند. پیچیدگی زمانی متعادل‌شده تولید جدول هلمن و سطح یک رابطه $T = (MX)^2 = 2^{d-l} \cdot \frac{N}{M^{d-l}}$ است، که به $2^{d-l} \cdot \frac{N}{M^{d-l}}$ می‌تواند کاهش یابد. در جدول ۴، مبادله‌های قبلی ارائه شده در (۵)، (۸) با این مبادله جدید برای $K=4, 8$ و برای دو مقدار حافظه خاص مقایسه شده‌اند. بدیهی است، پیچیدگی زمانی الگوریتم جدید برای همان مقدار حافظه موجود به‌طور قابل‌توجهی کوچک‌تر است [9].

۴- پیدا کردن برخوردهای کامل

روش اساسی پیدا کردن برخوردها یا نزدیک‌برخوردها در روش عمومی، محاسبه تابع چکیده‌ساز به تعداد زیاد با ورودی تصادفی است. بعد از i محاسبه تابع چکیده‌ساز، می‌توان $i(i-1)/2$ جفت را آزمایش کرد و این اثر تولد، اجازه پیدا کردن برخوردها را تنها با $O(2^{n/2})$ محاسبه تابع چکیده‌ساز می‌دهد. به‌طور دقیق‌تر، تعداد محاسبه مورد نیاز $\sqrt{\pi/2} \cdot 2^{n/2}$ است.

(جدول ۲): مقایسه معادله‌ها، برای سادگی، ضریب ثابت برای N نادیده گرفته شده است [9].

روش	M	T	پارامترهای دیگر
برنشتین و همکاران (۱)	$\frac{n}{2^4}$	$\frac{n}{2^2}$	-
$(T, M^2 = N)$	$\frac{n}{2^6}$	$\frac{2n}{2^3}$	-
الگوریتم پیشنهاد شده	$\frac{n}{2^4}$	$\frac{3n}{2^8}$	$X=2^{\frac{n}{8}}, I=\frac{n}{2}$

$(d-1)m$ بیت ذخیره‌شده وجود دارد؛ طوری که هیچ برخورد دیگر M مورد نیاز نیست، و تنها یک‌بار، در مجموع تا $(d-1)m$ بیت می‌تواند صفر باشند. با تنظیم $(d-1)m=n$ و در نتیجه الگوریتم K -درخت راه‌حلی برای مسئله K -فهرست را پیدا کردند. با این حال اگر اندازه حافظه محدود باشد (به‌عنوان مثال $\frac{n}{d+1} \ll m$) الگوریتم K -درخت تنها قسمتی از $(d-1)m$ بیت را به صفر می‌تواند اعمال کند. این الگوریتم جایگزینی سطح یک با تولید برخورد جدول هلمن و همان روش الگوریتم K -درخت از سطح دو به سطح D را انجام می‌دهد. برای پیدا کردن راه‌حل مورد نیاز پس از رسیدن به سطح D ، در سطح یک ارجاع به بیت بیشتر است. اجازه دهید تعداد بیت‌های متصل در سطح یک، L باشد. پس از سطح نخست 2^{d-1} لیست، هر کدام $M = 2^m$ عنصر دارند. به‌طور مشابه، بعد از رسیدن به سطح L برای $1+i, 2, 3, \dots, d-1$ فهرست شامل M عنصر با $l+(i-1)m$ بیت ذخیره‌شده وجود دارد. پس از رسیدن به سطح D ، یک عنصر با $L + DM$ بیت صفر خواهند داشت؛ بنابراین $L + DM = N$ (به‌عنوان مثال $l=n-dm$) را برای به‌دست‌آوردن دست‌کم یک راه‌حل در همه n بیت تنظیم کردند. در جدول (۱)، تعداد بیت‌های ذخیره‌شده از K -درخت و الگوریتم ارائه‌شده مقایسه شده‌اند.

(جدول ۱): مقایسه تعداد بیت‌های محصور بین K -درخت و

الگوریتم ارائه‌شده [9].

# لیست‌ها	# بیت‌های محصور	
	الگوریتم ارائه‌شده	الگوریتم K -درخت
سطح ۱ 2^{d-1}	l	m
سطح i $2^{d-i} (i=2, \dots, d-1)$	$l + (i-1)m$	im
سطح d ۱	$l + dm$	$(d+1)m$

از وضعیت $l=n-dm$ و پارامترهای K و M عملکرد کاهش f_1 را برای جدول هلمن می‌توان تعیین کرد. M زنجیره به‌طول X را ایجاد و تنها مقادیر نخست و آخر از هر زنجیر را در جدول هلمن موسوم به T_{pre} ذخیره کردند. هنگامی که T_{pre} ساخته شد، یک برخورد جزئی l بیتی را با

تشخیص داد. (این روش‌ها اغلب توسط حافظه مورد نیاز متفاوت در $O(\cdot)$ بین ۱ و ۳ است).

در این کار بر روش تمایز نقطه، به دلیل آن که به‌طور مؤثر می‌تواند موازی شود و هم‌چنین بر روی مسائل با پیچیدگی تاحدودی بزرگ تمرکز شده است. پیچیدگی پیدا کردن برخورد با استفاده از تمایز نقطه در جزئیات توسط ون اورشات^۵ [15] مورد تجزیه و تحلیل قرار گرفت. گام اصلی الگوریتم محاسبه زنجیره‌ای از تکرار، با شروع از یک نقطه تصادفی و توقف زمانی که به یک نقطه متمایز با ویژگی‌هایی قابل شناخت رسیده است (مانند تعداد صفرهای مقدم است). در این الگوریتم از یک جدول برای ذخیره M زنجیره (به‌عنوان مثال نقاط شروع و پایان) استفاده می‌شود و هنگامی که نقطه پایان دو مرتبه دیده شود، به احتمال زیاد یک برخورد به دست آمده است. تجزیه و تحلیل ون اورشات دو شرایط مختلف را (بسته به i تعداد برخوردهای مورد انتظار) در نظر می‌گیرد. یک پارامتر مهم در تجزیه و تحلیل نسبت نقطه مشخص شده θ است.

۴-۲- پیدا کردن تعداد کمی از برخورد به‌عنوان مثال $M \ll i$

اگر به اندازه کافی حافظه برای ذخیره تمام زنجیره‌ها موجود باشد، پیدا کردن i برخورد پس از یک حجم کار از $\theta(\sqrt{2^n i})$ را (از آنجایی که $\theta(\sqrt{2^n i})$ جفت نقاط را پوشش می‌دهد) می‌توان انتظار داشت. به‌طور دقیق‌تر، پیچیدگی داده شده توسط ون اورشات $C_{small} = \sqrt{\pi/2} \cdot \sqrt{2^n i} + 2.5i/\theta$ است.

ویژگی‌های تمایز انتخاب شده است؛ پس در نهایت تمام حافظه استفاده خواهد شد؛ اما سعی بر جلوگیری از دوباره نوشتن چرخه‌ها است، پس از $\theta = M/C_{small}$ استفاده می‌شود. که $C_{small} = \sqrt{\pi/2} \cdot \sqrt{2^n i} / (1 - 2.5i/M)$ اگر $i \ll M$ باشد، C_{small} به صورت زیر می‌شود:

$$C_{small} = \sqrt{\pi/2} \cdot \sqrt{2^n i}, \quad (9)$$

یک عامل تسریع از \sqrt{i} نسبت به پیدا کردن i برخورد به‌طور مستقل وجود دارد.

	$(T, M^2 = N)$	$\frac{n}{2^6}$	$\frac{5n}{2^{12}}$	$X=2^4, I=\frac{2n}{3}$
K=8	برنشتین و همکاران (۱)	$\frac{n}{2^5}$	$\frac{2n}{2^5}$	-
	$(T, M^3 = N)$	$\frac{n}{2^6}$	$\frac{n}{2^2}$	-
	برنشتین و همکاران (۳)	$\frac{n}{2^5}$	$\frac{2n}{2^5}$	-
	$(T^2, M = N)$	$\frac{n}{2^6}$	$\frac{5n}{2^{12}}$	-
	الگوریتم پیشنهاد شده	$\frac{n}{2^5}$	$\frac{3n}{2^{10}}$	$X=2^{10}, I=\frac{2n}{5}$
	$(T^2, M^2 = N)$	$\frac{n}{2^6}$	$\frac{n}{2^3}$	$X=2^6, I=\frac{n}{2}$

هنگام جستجو برای برخوردهای کامل، به جای مقایسه هر خروجی جدید با همه خروجی‌های پیشین آن (که نیاز به $\Omega(2^{2n})$ مقایسه دارد)، یک فهرست از تمام خروجی‌ها می‌توان تهیه و فهرست در زمان $O(n2^n)$ را دسته‌بندی و یا از یک جدول چکیده برای کاهش تعداد مقایسه‌ها به $O(2^n)$ استفاده کرد [10].

۴-۱- الگوریتم‌های کم حافظه^۱

حتی اگر از پیچیدگی محاسبه صرف‌نظر شود، پیچیدگی حافظه این روش ساده، آن را غیرعملی خواهد کرد. چندین کار نشان دادند که برخوردها می‌توانند با حافظه کم یا هیچ حافظه‌ای، با افزایش کوچک در پیچیدگی زمان پیدا شوند. ایده اصلی توسط پولارد^۲ با عنوان الگوریتم RHO برای فاکتورگیری و الگوریتم گسسته معرفی و بعد به جستجوی برخورد تعمیم داده شد. تابع چکیده‌ساز ابتدا از $\{0,1\}^*$ به $\{0,1\}^n$ به $\{0,1\}^n \rightarrow \{0,1\}^n$ محدود می‌شود، به طوری که آن را می‌توان تکرار کرد. پس از چند مرحله، زنجیره‌ای از تکرار به یک چرخه رسیده و نموداری به شکل حرف ρ یونانی خواهد داشت. به‌طور متوسط، حلقه دارای طول $O(2^{n/2})$ و پس از $O(2^{n/2})$ مرحله به دست آمده است. نقطه‌ای که دم ρ با چرخه ملاقات می‌کند، برخوردی در تابع چکیده‌ساز است. آن را در زمان $O(2^{n/2})$ با حافظه کم و یا هیچ حافظه‌ای با استفاده از روش‌های مختلف تشخیص چرخه، مانند الگوریتم فلویید^۳ [11]، الگوریتم برنت^۴ [12]، با استفاده از تمایز نقاط [13] یا چند روش دیگر [14] می‌توان

¹ Memory-less algorithms

² Pollard

³ Floyd's algorithm

⁴ Brent's algorithm

⁵ Van Oorschot

$$B_w(n) = B_w(n-1) + B_{w-1}(n-1), \quad (13)$$

قضیه ۱. نامساوی زیر نیز صدق می‌کند:

$$B_{w-1}(x) \leq \binom{x}{w} \frac{w}{x-2w+1}, \quad (14)$$

اثبات.

$$\begin{aligned} \frac{B_{w-1}(x)}{\binom{x}{w}} &= \frac{\binom{x}{w-1} + \binom{x}{w-2} + \binom{x}{w-3} + \dots}{\binom{x}{w}} \\ &= \frac{w}{x-w+1} + \frac{w(w-1)}{(x-w+1)(x-w+2)} + \dots \\ &\leq \frac{w}{x-w+1} + \left(\frac{w}{2-w+1}\right)^2 + \dots \\ &\leq \frac{\frac{w}{x-w+1}}{1-\frac{w}{x-w+1}} = \frac{w}{x-2w+1}, \end{aligned} \quad (15)$$

(با استفاده از مجموع سری هندسی)

۱-۵- الگوریتم حافظه-کامل^۱

در این الگوریتم تعداد محاسبات موردنیاز برای پیدا کردن یک نزدیک‌برخورد $i = \sqrt{\pi/2 \cdot 2^n / B_w(n)}$ است. هم‌چنین حد پایین تعداد ارزیابی چکیده موردنیاز برای هر الگوریتم نزدیک‌برخورد دست‌کم $\sqrt{\pi/2 \cdot 2^n / B_w(n)}$ محاسبه جهت دستیابی به w نزدیک‌برخورد با احتمال ناچیز است.

با این حال، این روش ساده نیاز به $B_w(n)$ پیچیدگی را با استفاده از الگوریتم مرتب‌سازی، جدول چکیده و یا زنجیره تکرار نمی‌توان کاهش داد. در واقع در هر اجرای عملی ضعف حساب می‌شود [10].

۲-۵- استفاده از برخورد در یک تابع چکیده

بریده‌شده

یک روش ساده، پیدا کردن برخوردها در نسخه بریده‌شده تابع چکیده‌ساز است. در راحت‌ترین روش، تابع چکیده‌ساز به $t = n-w$ بریده می‌شود و هر برخورد در نسخه بریده‌شده w نزدیک‌برخورد را برای تابع چکیده‌ساز کامل ایجاد می‌کند. جالب توجه است، زمانی که تابع چکیده‌ساز به t

¹ Memory-full algorithm

۳-۴- پیدا کردن تعداد زیادی از برخوردها

به‌عنوان مثال $M \gg i$

در این مورد، حافظه باید بازنویسی شود. آزمایش‌ها [15] نشان می‌دهند زمانی که حافظه پر است، پیچیدگی هر برخورد به‌طور تقریبی $2^n \theta / M + 2/\theta$ است. این پیچیدگی به کمینه $\sqrt{8 \cdot 2^n / M}$ برای $\sqrt{2M/2^n \theta}$ می‌رسد. به‌طور دقیق‌تر، ون اورشات آزمایش‌هایی برای تعیین ثابت‌های حقیقی انجام داده و پیچیدگی زیر حاصل شده است:

$$C = 5\sqrt{2^n / M} \cdot i, \quad (10)$$

هنگامی که $2.25\sqrt{M/2^n \theta}$ است. یک عامل تسریع از $\sqrt{M}/4$ نسبت به پیدا کردن i برخورد به‌طور مستقل وجود دارد.

حد جهانی. به‌طور کلی، یک کران بالا پیچیدگی که در هر دو شرایط کار می‌کند، با جمع دو عبارت می‌توان بیان کرد:

$$C \leq \left(\sqrt{\frac{\pi}{2}} + 5\sqrt{\frac{i}{M}}\right) \sqrt{2^n i}. \quad (11)$$

هنگامی که $M \gg i$ یا $M \gg i$ ، یک دوره ناچیز است، این عبارت به ترتیب معادل C_{small} یا C_{large} است. علاوه‌براین، به‌طور تجربی تأیید کردند، زمانی که $i \approx M$ این عبارت حد بالا و تاحدودی باریک است. در همه موارد یک افزایش سرعت خطی وقتی که از چند ماشین به‌موازات استفاده می‌شود، وجود دارد [10].

۵- نزدیک برخوردها

w نزدیک‌برخورد یک جفت پیام x و x' ، به‌صورت $h(x) \oplus h(x') \leq w$ است، که در آن $\| \cdot \|$ وزن همینگ است. ابتدا برخی از نتایج در ارتباط با فاصله همینگ معرفی می‌شود. تعریف ۱. اندازه یک توپ همینگ به شعاع w توسط $\{x \in \{0,1\}^n : \|x\| \leq w\}$ نشان می‌دهند.

ویژگی ۱. وجود دارد $B_w(n) = \sum_{i=0}^w \binom{n}{i}$ و ویژگی ۲. احتمال این که یک جفت x و x' تصادفی در w نزدیک‌برخورد نتیجه دهد برابر است با:

$$B_w(n) / 2^n, \quad (12)$$

ویژگی ۳. رابطه زیر برقرار است:

مقداری حافظه می‌تواند به‌طور قابل‌توجهی توسط عامل \sqrt{i} در صورتی که $M \gg i$ یا $\sqrt{M}/4$ و $M \ll i$ (همان‌طور که در بخش ۴ شرح داده شد) باشد، کاهش یابد. در ادامه، ایده موردنظر و مطالعه مقدار بهینه τ و پیچیدگی حمله، بسته به میزان حافظه در دسترس شرح داده می‌شود. در پیاده‌سازی عملی یک حمله نزدیک‌برخورد فرض می‌شود که مقداری از حافظه در دسترس است، که به‌طور قابل‌توجهی منجر به حمله‌های بهتر می‌تواند شود.

۱-۶- پیچیدگی

مطابق با شکل (۳) τ بیت از تابع چکیده‌ساز را برش داده و برای برخورد در $n - \tau$ بیت باقی‌مانده جستجو می‌کنند. برای هر برخورد $n - \tau$ بیتی، فاصله همینگ در τ بیت برش یافته محاسبه می‌شود. در نتیجه پیدا کردن یک w نزدیک‌برخورد پس از آزمایش $i = 2^{\tau} \cdot B_w(\tau)$ برخورد امکان‌پذیر است.

مشاهده می‌شود که $i(\tau)$ به‌طور یکنواخت با توجه به $\tau B_w(\tau-1) + B_w(\tau-1) < 2B_w(\tau-1)B_w$ می‌یابد. τ کوچک، تنها نیاز به تعداد کمی از برخوردها است؛ اما پیدا کردن برخوردها به دلیل تعداد زیاد بیت برش‌نیافته $n - \tau$ ، یک ارزیابی دقیق از پیچیدگی الگوریتم، بسته به مقادیر M و τ است، که از تجزیه و تحلیل ون اورشات [15] (که در بخش ۴ یادآوری شد) استفاده شده است.

$n - 2w - 1$ بیت کوتاه شود، یک برخورد t بیتی، یک w نزدیک‌برخورد تابع چکیده‌ساز کامل با احتمال $1/2$ را ایجاد خواهد کرد. این یک الگوریتم نزدیک‌برخورد با پیچیدگی موردانتظار $\sqrt{\pi/2} 2^{(n-2w)/2}$ با استفاده از مقدار کمی حافظه برای نقاط متمایز است. که به‌صورت گرافیکی شکل (۲) آن را می‌توان نشان داد.

به‌طور کلی، τ بیت را می‌توان برش داد؛ برخورد در یک تابع $n - \tau$ بیتی را پیدا و وزن همینگ τ بیت کوتاه‌شده را بررسی کرد و در نهایت w نزدیک‌برخورد را با احتمال $B_w(\tau)/2^{\tau}$ به‌دست آورد. مقدار بهینه τ را نیز با ارزیابی پیچیدگی برای تمام گزینه‌های τ می‌توان پیدا کرد.

۶- بده‌بستان زمان-حافظه با روش کوتاه‌سازی

نخستین الگوریتم تعمیم ساده روش مبتنی بر کوتاه‌سازی در بخش ۵ است. مشاهده شد که اگر τ بیت با $1 > 2w$ برش داده شود، احتمال این که یک برخورد در تابع کوتاه‌شده یک نزدیک‌برخورد در تابع چکیده‌ساز کامل باشد، به‌سرعت کاهش می‌یابد و نیاز به پیدا کردن برخوردهای بسیاری دارد. در یک رویکرد حقیقی حافظه کم‌تر، پیدا کردن i چنین برخورد نیاز به $\sqrt{\pi/2} \cdot i \sqrt{2^{n-\tau}}$ محاسبات دارد و برای به‌دست‌آوردن توسط روش برش‌دادن بیش از $2w - 1$ بیت، کم‌تر وجود دارد. با این حال، با



(شکل-۲): شکل گرافیکی یک الگوریتم پیدا کردن نزدیک-برخورد [10].



(شکل-۳): شکل گرافیکی الگوریتم پیدا کردن نزدیک-برخورد با استفاده از حافظه کم [10].

۲-۶- پیدا کردن پارامترهای بهینه

در جهت پیدا کردن خصوصیات جبری τ بهینه، تجزیه و تحلیل بخش ۴ را دنبال کرده و دو مورد برای پیچیدگی،

بسته به رابطه بین M و i در نظر گرفتند. با در نظر گرفتن تعداد کمی از بیت برش‌یافته و برخورد $M \ll 2^{\tau} \cdot B_w(\tau)$ پیچیدگی زیر به‌دست می‌آید:

برای این اندازه τ ، پیچیدگی به صورت زیر است:

$$C \approx 2^{n/2} / \sqrt{B_w(\tau)}, \quad (20)$$

که این پیچیدگی بیشتر از پیچیدگی بهینه به دست آمده توسط الگوریتم حافظه کامل $2^{n/2} / \sqrt{B_w(n)}$ است؛ اما برای بیش تر پارامترها از حد $2^{n/2} / \sqrt{B_w/2}(n)$ (که الگوریتم مبتنی بر کد پوششی را محدود می کند) بهتر است.

τ بهینه در عمل. برای n و m داده شده، تخمین

بهتری از τ بهینه می توان پیدا کرد. از حد بالای (21) که حد بالای پیچیدگی زیر را می دهد، استفاده شده است:

$$C \leq C_{small} + C_{lg} = \left(\sqrt{\frac{\pi}{2}} + 5 \sqrt{\frac{2^{\tau/B_w(\tau)}}{M}} \right) \cdot \sqrt{\frac{2^n}{B_w(\tau)}}, \quad (21)$$

برای پیدا کردن یک بدهستان خوب، این حد را برای تمام مقادیر τ محاسبه و از مقداری τ که پایین ترین حد را می دهد، استفاده کردند. در مقاله [10] نمونه اعمال این حمله بر MD5 نشان داده است.

۷- نتایج

در این مقاله، حمله TMTO و روش های بهبود این حمله که تاکنون ارائه شده اند، معرفی و همچنین، الگوریتم عمومی جهت پیدا کردن نزدیک برخورد ها معرفی شد، که هردو الگوریتم پیشین مبتنی بر کوتاه سازی و مبتنی بر کدهای پوششی را تعمیم می دهد. علی رغم کارهای پیشین، هدف این الگوریتم مطالعه TMTO بوده است و در عمل قابل پیاده سازی است. لورنت ثابت کرد که با در نظر گرفتن یک مقدار حافظه عملی، با تغییر دادن پارامترها بهبودی در عملکرد این الگوریتم می توان مشاهده کرد. این الگوریتم از الگوریتم جستجو موازی برخورد اورشات ایده گرفته است، که با حافظه کم در زمان مناسبی کمتر از $i \sqrt{2^n}$ برخورد را می توان پیدا کرد.

۸- مراجع

[1] Su, B. Wu. W. Dong. L, *Near-collisions on the reduced-round compression functions of Skein and BLAKE*, in *Cryptology and Network Security*. 2010, Springer. p. 124-139.

$$= \sqrt{\pi/2} \cdot \sqrt{2^{n-\tau} \cdot 2^{\tau/B_w(\tau)}} = C_{small} \sqrt{\pi/2} \cdot \frac{2^{n/2}}{\sqrt{B_w(\tau)}}, \quad (16)$$

وقتی که τ افزایش می یابد، این عبارت نیز افزایش پیدا می کند. با تعداد زیاد بیت برش یافته و تعداد زیاد برخورد ها $M \gg 2^{\tau} \cdot B_w(\tau)$ پیچیدگی برابر است با:

$$C_{large} = \frac{5 \sqrt{2^{n-\tau}/M} \cdot 2^{\tau}}{B_w(\tau)} = \frac{5 \cdot 2^{n/2+\tau/2}}{B_w(\tau) \sqrt{M}}, \quad (17)$$

با مطالعه تغییرات C_{large} :

$$(\tau - 1) \leq C_{large}(\tau) C_{large} \quad (18)$$

$$\begin{aligned} \frac{C_{large}(\tau - 1)}{C_{large}(\tau)} &\leq 1 \\ \frac{B_w(\tau)}{B_w(\tau - 1)} &\leq \sqrt{2} \\ \frac{B_w(\tau - 1) + B_{w-1}(\tau - 1)}{B_w(\tau - 1)} &\leq \sqrt{2} \\ \frac{B_{w-1}(\tau - 1)}{B_w(\tau - 1)} &\leq \sqrt{2} - 1 \\ \frac{B_w(\tau - 1)}{B_{w-1}(\tau - 1)} &\geq \sqrt{2} + 1 \\ \frac{\binom{\tau-1}{w} + B_{w-1}(\tau - 1)}{B_{w-1}(\tau - 1)} &\geq \sqrt{2} + 1 \\ \frac{\binom{\tau-1}{w}}{B_{w-1}(\tau-1)} &\geq \sqrt{2}, \end{aligned}$$

با استفاده از قضیه ۱. $\binom{\tau-1}{w} / B_{w-1}(\tau - 1) \geq$

هنگامی که $\tau \geq (\sqrt{2} + 2)w$ نتیجه $\binom{\tau-1}{w} / B_{w-1}(\tau - 1) \geq \sqrt{2}$ را افزایش می دهد. توجه داشته باشید که این فرمول تنها زمانی که $M \ll 2^{\tau} / B_w(\tau)$ یعنی برای مقادیر τ و این فرض که $\tau \geq (\sqrt{2} + 2)w$ صدق می کند. در این حوزه برای مقادیر مناسب پارامترها درست خواهد بود. به ویژه در $M > 2^{24}$ و $w < 48$ نیز درست است.

برای $M \approx 2^{\tau} / B_w(\tau)$ دو بیان تا رسیدن به مقدار ثابت کوچک برابر هستند. مشابه بخش ۴، در واقع پیچیدگی پیوسته است.

τ بهینه. زمانی که τ کوچک باشد، یعنی $M \ll 2^{\tau} / B_w(\tau)$ پیچیدگی با τ کاهش می یابد؛ اما زمانی که τ بزرگ باشد، یعنی $M \gg 2^{\tau} / B_w(\tau)$ با افزایش τ افزایش می یابد. با انتخاب بهینه τ ، رابطه زیر ایجاد می شود:

$$M \approx 2^{\tau} / B_w(\tau) \quad (19)$$



زهرا ذوالفقاری تحصیلات خود را در مقطع کارشناسی ارشد مهندسی مخابرات گرایش رمز در سال ۱۳۹۵ در دانشگاه تربیت دبیر شهید رجایی به پایان رساند. زمینه پژوهشی مورد علاقه وی تحلیل و بررسی امنیتی توابع چکیده ساز و الگوریتم های رمز است.



نصور باقری، دانشیار دانشگاه تربیت دبیر شهید رجایی، کارشناسی خود را در دانشگاه مازندارن و دوره کارشناسی ارشد و دکترا را در دانشگاه علم و صنعت ایران در مهندسی الکترونیک به پایان رساند. مقالات متعددی در زمینه رمزشناسی توسط ایشان، در مجلات و همایشهای ملی و بین‌المللی، ارائه شده است. در حال حاضر زمینه پژوهشی وی رمزنگاری و امنیت اطلاعات است.

- [2] Jean, J. and P.-A. Fouque. *Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 Compression Function*. in *FSE*. 2011. Springer.
- [3] Leurent, G. and S.S. Thomsen. *Practical near-collisions on the compression function of BMW*. in *Fast Software Encryption*. 2011. Springer.
- [4] Lamberger, M. and V. Rijmen. *Optimal covering codes for finding near-collisions*. in *Selected Areas in Cryptography*. 2010. Springer.
- [5] Wagner, D. *A generalized birthday problem*. in *Annual International Cryptology Conference*. 2002. Springer.
- [6] Minder, L. and A. Sinclair, *The extended k-tree algorithm*. *Journal of cryptology*, 2012. **25**(2): p. 349-382.
- [7] Stoffelen, K. *Comparison of chain merge behaviour of TMTO methods*. 2013, BSc Ithesis, Radboud University of Netherlands.
- [8] Bernstein, D.J. *Better price-performance ratios for generalized birthday attacks*. in *Workshop Record of SHARCS*. 2007.
- [9] Nikolić, I. and Y. Sasaki. *Refinements of the k-tree Algorithm for the Generalized Birthday Problem*. in *International Conference on the Theory and Application of Cryptology and Information Security*. 2014. Springer.
- [10] Leurent, G. *Time-memory Trade-offs for Near-collisions*. in *Fast Software Encryption*. 2013. Springer.
- [11] Knuth, D.E., *Seminumerical Algorithms, volume 2 of The Art of Computer Programming*. Addison Wesley. Reading, MA, 1969.
- [12] Brent, R.P., *An improved Monte Carlo factorization algorithm*. BIT Numerical Mathematics, 1980. **20**(2): p. 176-184.
- [13] Quisquater, J.-J. and J.-P. Delescaille. *How easy is collision search. New results and applications to DES*. in *Advances in Cryptology—Crypto '89 Proceedings*. 1989. Springer.
- [14] Nivasch, G., *Cycle detection using a stack*. Information Processing Letters, 2004. **90**(3): p. 135-140.
- [15] Van Oorschot, P.C. and M.J. Wiener, *Parallel collision search with cryptanalytic applications*. *Journal of cryptology*, 1999. **12**(1): p. 1-28.

