معرفی حمله بده‌بستان زمان-حافظه (TMTO) بر توابع چکیده‌ساز

زهرا ذوالفقاری* و نصور 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 پیداکردن نزدیک- برخوردها در یک تابع چکیده‌ساز پرداخته شده است. با درنظرگرفتن 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. به‌تازگی، 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 قابل‌توجه پیچیدگی را می‌توان کاهش داد؛ در مرحله بعد، با استفاده از برخوردهای متعدد براساس جدول هلمن، بهبود شناخته‌شده‌ترین بده‌بستان زمان حافظه، برای K درخت ارائه شد. درنتیجه، منحنی بده‌بستان جدید $T^2.M^{\lg k-1} = k.N$ به‌دست آورده شد، با درنظرگرفتن $k = 4$ منحنی بده‌بستان به شکل $T^2.M = 4.N$ خواهد بود. در این مقاله، ابتدا روش‌های TMTO و سپس نحوه پیداکردن نزدیک-برخورد با استفاده از TMTO شرح داده می‌شود.

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

۱- مقدمه

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:

$$h:\{0,1\}^{*} \rightarrow \{0,1\}^{n}, \qquad (۱)$$

هنگامی‌که در زمینه رمزنگاری استفاده می‌شود، یک تابع چکیده‌ساز باید در برابر سه حمله اصلی مقاوم باشد:

حمله برخورد[1]: زمانی که h داده شده است، بتوان $x \neq x'$ پیدا کرد که $h(x)=h(x')$.

حمله پیش‌تصویر[2]: زمانی که h و y داده شده است، بتوان x پیدا کرد که $h(x)=y$.

حمله پیش‌تصویر دوم[3]: 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 پیداکردن برخورد است. اغلب نه‌تنها در پیداکردن برخورد برای توابع چکیده‌ساز بلکه در مسائل دیگر رمزنگاری نیز مطرح است. طی چندین دهه، پیداکردن برخورد به‌طور گسترده مورد مطالعه قرار گرفته است. علاوه‌بر این مسئله و همراه با پیداکردن 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؛ درحالی‌که حمله‌های پیش‌تصویر یا پیش‌تصویر دوم دارای پیچیدگی 2^{n} هستند: که نیازمندی‌های امنیتی یک تابع چکیده‌ساز n بیتی را تعریف

[1] Collision attack
[2] Priemage attack
[3] Second-priemage attack

* نویسندهٔ عهده‌دار مکاتبات

[Downloaded from monadi.isc.org.ir on 2026-11-09]

می‌کنند. به‌طورکلی، انتظار می‌رود که یک تابع چکیده‌ساز مانند یک تابع تصادفی رفتارکند. 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 درنتیجه 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 تاحدودی 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 $2^{\frac{n}{2}}$ حافظه در دسترس دارد. برای حل این مشکل، لمبرگر [1] و همکاران یک روش با حافظه کمتر مبتنی بر کدهای پوششی [4]، با یک پیچیدگی بین $2^{\frac{n}{2}}$ 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 رمزنگاری بسیار مهم تلقی می‌شود؛ اما بیش از یک دهه پس از انتشار آن بهبود قابل توجهی در الگوریتم K-درخت و در دیگر الگوریتم های 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 زمانی‌که 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.

۲-۱- الگوریتم‌های جستجوی جامع و TMTO

فرض کنید، مهاجم قصد رمز گشایی‌کردن پیام رمزگذاری شده 'm دارد. هدف محاسبه m برای f (m)=m' است که در آن f می تواند یک تابع چکیده ساز، رمز قالبی[2] و یا یک رمز جریایی[3] 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 nبیتی، به‌طورمعمول $N=2^n$ کلید ممکن وجود دارد. هنگامی‌که 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<m,f(m)> ذخیره کرد. این مرحله، برون‌خط حمله نامیده می‌شود. زمانی‌که یک نفر بخواهد یک پیام

[1] Lamberger

[2] Block cipher

[3] Stream cipher

[Downloaded from monadi.isc.org.ir on 2026-11-09]

رمزشده را بشکند، در طول مرحلهٔ برخط تنها باید آن را در جدول برای پیدا کردن کلید جستجو کند؛ که کاری غیرمفید است و نیاز به مقدار زیادی از فضای حافظه و زمان طولانی برای ایجاد جدول‌ها دارد. برای حل این مشکل بده بستان زمان-حافظه پیشنهاد شد، که در آن می‌توان سرعت یک الگوریتم را درازای هزینه حافظه بیشتر، افزایش داد و بالعکس.

هلمن[1]

در سال 1981، هلمن حمله TMTO احتمالی خود را معرفی کرد که، جستجوی جامع هر سامانه رمزنگاری با Nکلید ممکن را پس از یک پیش‌مجاسبه با کاهش پیچیدگی $N^{2/3}$ عملیات، ولی با صرف $N^{2/3}$ کلمه حافظه می‌تواند اجرا کند. این کار را با محاسبه ماتریس m×t به‌صورت نشان‌داده‌شده در زیر می‌توان انجام داد. این حمله متشکل است از m زنجیره تکرارشونده به‌طول t، که در آن تنها لازم است، نقطه نخست و آخر هر زنجیره در یک جدول با جفت <x,f_i^t(x)> ذخیره شوند. با مطالعهٔ مقاله هلمن، به‌طورمعمول m و t به‌صورت $mt^2=N$ انتخاب می‌شوند.

با این حال، ماتریس m×t از $mt^2=N$ تبعیت می‌کند که تنها $1/t$ -ام از فضای جستجو را پوشش می‌دهد. به همین دلیل است که این ماتریس‌ها را محاسبه و هرکدام بخش‌های مختلفی از فضای جستجو را اشغال می‌کنند. هر کدام از این ماتریس‌ها از یک f_i متفاوت استفاده می‌کنند، که به‌صورت $f_i=h_i \circ f$ تعریف شده (h_i خروجی اصلاح شده و برای هر i متفاوت است). یک نمونه $h_i(x) = x \oplus i$ است. هدف استفاده از توابع مختلف f_i کاهش مقدار زنجیره‌های تکراری بین جدول‌ها است[7].

$$\begin{array}{l} x_0 \to f_i(x_0) \to f_i(f_i(x_0)) \to \cdots \to f_i^t(x_0) \\ x_1 \to f_i(x_1) \to f_i(f_i(x_1)) \to \cdots \to f_i^t(x_1) \\ x_2 \to f_i(x_2) \to f_i(f_i(x_2)) \to \cdots \to f_i^t(x_2) \\ \vdots \quad\quad \vdots \quad\quad\quad \vdots \quad\quad\quad \vdots \\ x_m \to f_i(x_m) \to f_i(f_i(x_m)) \to \cdots \to f_i^t(x_m), \end{array} \qquad (۲)$$

در مرحلهٔ برخط مهاجم f_i را برای بیشینه زمان t بر روی متن رمزی که خودش داده است، تکرار و هر مرحله بررسی می‌کند، که نتیجه در جدول مشاهده می‌شود یا نه. اگر با $f_i^t(x_j)$ منطبق باشد، مهاجم f_i را می‌تواند تکرار تا متن رمزشدهٔ خود را پیدا کند. به‌عبارتی، پیش‌تصویر متن رمزشدهٔ خود را نیز می‌تواند پیدا کند.

[1] Hellman

۲-۲- نقاط متمایز[2]

بعدا، ریوست[3] بهبودیافته روش هلمن را پیشنهاد کرد؛ که هدف آن کاهش تعداد دسترسی به حافظه و بنابراین سرعت‌بخشیدن جستجوی جامع است؛ درحالی‌که دسترسی به حافظه به‌طورمعمول زمان بیشتری از محاسبات f_i را صرف می‌کند. در این روش، شرط خاتمه هر زنجیره، رسیدن به یک نقطه پایانی DP به‌عنوان نقطه‌ای خاص با خاصیت ویژه (مانند K بیت نخست صفر باشد) می‌شود. اکنون تنها مقادیر خروجی تابع که دارای این خاصیت هستند، باید در لوح سخت حافظه مورد جستجو قرار بگیرد. بار دیگر، ماتریسa محاسبه و یک تابع $f_i=h_i \circ f$ بر زنجیره‌های با طول متغیر تکرار می‌شود. نقاط پایانی دارای پیشوند K بیت صفر ذخیره می‌شوند [7].

$$\begin{array}{l} x_0 \to \cdots \to f_i^p(x_0) = DP_0 \\ x_1 \to \cdots \to \cdots \to \cdots \to f_i^q(x_1) = DP_1 \\ x_2 \to \cdots \to \cdots \to f_i^r(x_2) = DP_2 \\ x_m \to \cdots \to f_i^*(x_m) = DP_m \ , \end{array} \qquad (۳)$$

از آنجا که ممکن است، مدت‌زمانی را (یا حتی برای همیشه درمورد چرخه‌ها) قبل از رسیدن به یک نقطه متمایز صرف شود، یک T_{MAX} ثابت به‌عنوان بیشینهٔ طول زنجیره در نظر می‌گیرند. اگر یک زنجیره قبل از رسیدن به نقطه متمایز به T_{MAX} برسد، زنجیره‌ای بی‌استفاده است.

۲-۳- جداول رنگین‌کمان[4]

برای یک مدت طولانی، روش ریوست تنها بهبود حمله هلمن TMTO بوده است؛ تا اینکه سال ۲۰۰۳ بهبود جدید الگوریتم‌های اوکسلین [5] با نام جداول رنگین‌کمان پیدا شد. یکی از نواقص الگوریتم هلمن ایجاد برخورد در یک جدول با ادغام زنجیره‌ها بود. اوکسلین به جای درخواست مجدد به تابع رمزنگاری با اضافه‌کردن یک خروجی اصلاح‌شده به هر مرحله در زنجیره، این مشکل را بهبود داد. به این‌ترتیب، هر ستونf_i خود را اعمال می‌کند. در حال‌حاضر، با ادغام زنجیره‌ها یک برخورد ایجاد نمی‌شود، مگر اینکه برخورد در همان ستون اتفاق بیفتد. این موضوع باید از یک بخش بزرگی از ادغام زنجیره‌ای جلوگیری کند. در این روش، تنها یک ماتریس بزرگ m×t محاسبه شده است؛ پس لازم نیست

[2] Distinguished points
[3] Rivest
[4] Rainbow tables
[5] Oechslin

[Downloaded from monadi.isc.org.ir on 2026-11-09]

ماتریس‌های متفاوت محاسبه شوند. تنها نقاط نخست و آخر هر زنجیره ذخیره می‌شوند [7].

$$\begin{array}{l} x_0 \to f_0(x_0) \to f_1(f_0(x_0)) \to \cdots \to \\ f_t(f_{t-1}(\cdots f_0(x_0)\cdots)) \\ x_1 \to f_0(x_1) \to f_1(f_0(x_1)) \to \cdots \to \\ f_t(f_{t-1}(\cdots f_0(x_1)\cdots)) \\ x_2 \to f_0(x_2) \to f_1(f_0(x_2)) \to \cdots \to \\ f_t(f_{t-1}(\cdots f_0(x_2)\cdots)) \\ \vdots \quad \vdots \quad \vdots \\ x_m \to f_0(x_m) \to f_1(f_0(x_m)) \to \cdots \to \\ f_t(f_{t-1}(\cdots f_0(x_m)\cdots)), \end{array} \qquad (۴)$$

۳- بهبود بده‌بستان زمان-حافظه

در برنامه‌های کاربردی، به‌طورمعمول عناصر فهرست L_i خروجی توابع f_i می‌باشند؛ درنتیجه GBP اغلب به‌صورت زیر فرموله می‌شود:

مسئله ۱. با توجه به توابع غیرمعکوس $f_1, \dots, f_k: \{0,1\}^{n'} \to \{0,1\}^n, n' \geq n$ پیدا کنید $y_1, \dots, y_k \in \{0,1\}^{n'}$ به‌گونه‌ای که $f_1(y_1) \oplus f_2(y_2) \oplus \dots \oplus f_k(y_k) = 0$.

در برخی از برنامه‌های کاربردی، تمام توابع یکسان هستند و مشکل پیداکردن ورودی مستقل است؛

مسئله ۲. با توجه به تابع غیرمعکوس $f: \{0,1\}^{n'} \to \{0,1\}^n, n' \geq n$ پیدا کنید $y_1, \dots, y_k \in \{0,1\}^{n'}$ مستقل را به‌گونه‌ای که $f_1(y_1) \oplus f_2(y_2) \oplus \dots \oplus f_k(y_k) = 0$.

هر دو مسئلهٔ بالا باعث افزایش احتمال بده‌بستان زمان حافظه می‌شوند(یعنی کاهش پیچیدگی حافظه الگوریتم K–درخت به‌ازای هزینه زمان)؛ بنابراین بده‌بستان زمان حافظه برای GBP بررسی خواهد شد که در مسئله ۲ تعریف شده است. توجه داشته باشید که K–درخت به‌صورتی فرض می‌شود که هر دو زمان و حافظه اندازه‌های یکسان دارد؛ به‌عنوان مثال $T = M = O(k.2^{\frac{n}{\lg k+1}})$. برنشتاین و همکاران [8] الگوریتم K–درخت را در محیط‌های حافظه محدود و پیشنهاد چند بده‌بستان بررسی کردند. رویکرد اصلی آنها برای حل این مشکل، استفاده K-فهرست در کمتر از n بیت است. فرض کنید $M=2^m$، که $M < 2^{\frac{n}{\lg k+1}}$ است؛ سپس، مشکل K–فهرست $\bar{n} = m(\lg k + 1)$بیت (به‌جای n بیت) را به‌راحتی با استفاده از الگوریتم K–درخت می‌توان حل کرد. نخستین ایده بده‌بستان انجام یک پیش‌محاسبه (یا prefiltration) است؛ به‌صورتی که تمام نوشته‌های در هر فهرست، دارای مقدار صفر در $n - \bar{n}$ بیت پر ارزش خود هستند. برای$\bar{n}$ بیت باقی‌مانده کم‌ارزش، آن‌ها الگوریتم K–درخت را اعمال و در نتیجه یک راه حل برای تمام n بیت پیدا کردند. پیچیدگی زمانی مجموع هزینه‌ها برای پیش‌محاسبه و حل الگوریتم K–درخت، $k.(2^{n-\bar{n}}.2^m + 2^m) \approx k.2^{n-m\lg k}$ است. بنابراین بده‌بستان به‌صورت زیر تعریف می‌شود:

$$T.M^{\lg k} = k.N, \qquad (۵)$$

ایده دوم آنها مشابه ایده نخست است؛ اما از پیش‌محاسبه استفاده نمی‌کند. در این روش، الگوریتم K-درخت را بر $\bar{n} = m(\lg k + 1)$ بیت اعمال کردند، تا مقدار $n - \bar{n}$ بیت باقی‌مانده به‌طوراحتمالی صفر شود. در کل $2^{n-\bar{n}}$ تکرار K–درخت وجود دارد؛ سپس پیچیدگی زمانی $T = k.2^m.2^{n-\bar{n}} = k.2^{n-m\lg k}$ می‌شود، که بده‌بستان مشابه مورد قبلی فراهم می‌کند. یعنی،

$$T.M^{\lg k} = k.N, \qquad (۶)$$

ایده سوم مبتنی بر کاهش N ، اما روشی پیشرفته‌تر است. فرض کنید $f_1 = f_2, f_3 = f_4, \dots$ ، یعنی توابع دوبه‌دو یکسان باشند. مسئله K–فهرست به‌عنوان دو $\frac{k}{2}$ مسئله جداگانه در نظر گرفته می‌شود، ابتدا شامل توابع $f_1, f_3, \dots$ ، درحالی‌که دومین مسئله $f_2, f_4, \dots$ (اگر مقدار حافظه در دسترس 2^m باشد) است؛ سپس امکان حل هر یک از این $\frac{k}{2}$ فهرست مسئله تا $\bar{n} = m(\lg k + 1) = m.\lg k$ بیت وجود دارد. با بالابردن دو $\frac{k}{2}$–فهرست به K–فهرست، $n - \bar{n}$ بیت باقی‌مانده با استفاده از الگوریتم جستجوی برخورد حافظه، می‌توانند صفر باشند. بنابراین پیچیدگی زمان $T = \frac{k}{2}.2^{\frac{n-\bar{n}}{2}}.2^m = \frac{k}{2}.2^{\frac{n}{2}-m(\frac{\lg k}{2}-1)}$ است و منحنی بده‌بستان به‌صورت زیر تعریف می‌شود:

$$T.M^{\frac{\lg}{2}-1} = \frac{k}{2}.N^{\frac{1}{2}}, \qquad (۷)$$

که تبدیل شده است به:

$$T^2.M^{\lg k-2} = \frac{k^2}{4}.N, \qquad (۸)$$

از آنجا که این روش مسئله $\frac{k}{2}$– فهرست را حل می‌کند، برای حالت K> 4 قابل اعمال است. هم‌چنین در حالتی که $M < 2^{\frac{n}{\lg k+2}}$ باشد، (۸) بده‌بستان بهتری را نسبت به (۵) $M > 2^{\frac{n}{\lg k+2}}$ فراهم می‌کند.

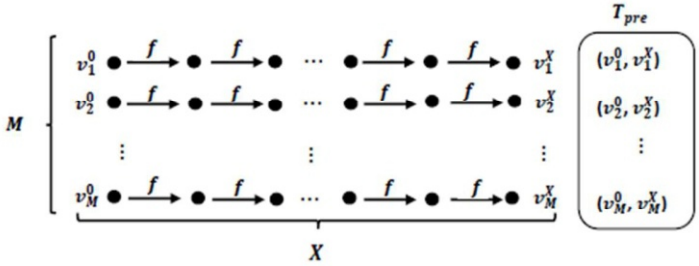
[Downloaded from monadi.isc.org.ir on 2026-11-09]

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 زمانی‌که 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، پیداکردن 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 $f:\{0,1\}^* \to \{0,1\}^n$ یک ورودی دلخواه و تابع خروجی n بیتی باشد، $N=2^n$ و $M=2^m$ مقدار حافظه در دسترس است. هنگامی که پیش‌محاسبه با ارزیابی MX از f انجام‌شود، هزینه تولید برخورد جدید برای f ، $\frac{N}{MX}$ به ازای هر برخورد است.

این روش به‌شرح زیر کار می‌کند:

انتخاب M مقدار متمایز $v_i^0 \in \{0,1\}^n$، که $i=1,2,\ldots,M$. 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 $i=1,\ldots,M, j=1,\ldots,X$ 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.



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

توجه داشته باشید، حتی اگر مقادیر MX در تمام زنجیره‌ها وجود داشته باشد، فقط مقادیر 2Mدر T_{pre} ذخیره می‌شوند. هنگامی‌که T_{pre} ساخته شود، برای تولید یک برخورد، با یک نقطه تصادفی شروع کرده و یک زنجیره‌ای به‌طول $\frac{N}{MX}$ 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 زنجیرهای 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 پیداکردن نه تنها یک برخورد، بلکه برخوردهای متعدد می‌توان استفاده کرد.

ژو[1] و لوکس[2] از این روش برای پیداکردن 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 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 L <n) 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. به‌طورکلی، 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 $i = 2,3, \ldots, d-1$ ، 2^{d-i} حاوی M عنصر با im بیت ذخیره‌شده تولید می‌شوند. در آخرین سطح D دو فهرست حاوی M عنصر با

[1] Joux
[2] Lucks

[Downloaded from monadi.isc.org.ir on 2026-11-09]

(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 می‌توانند صفر باشند. با تنظیم (d-1)m=n و درنتیجه الگوریتم K-درخت راه‌حلی برای مسئله K-فهرست را پیدا کردند. با این حال اگر اندازه حافظه محدود باشد (به‌عنوان مثال $m \ll \frac{n}{d+1}$) 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 پیداکردن راه‌حل موردنیاز 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 $i = 2,3, \dots, d-1$ فهرست شامل M عنصر با l+(i-1)m بیت ذخیره‌شده وجود دارد. پس از رسیدن به سطح نهاییD، 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 K-درخت	الگوریتم ارائه‌شده
سطح ۱ 2^{d-1}	m	l
سطح i 2^{d-i} $(i=2,\dots,d-1)$	im	$l+(i-1)m$
سطح d ۱	$(d+1)m$	$l+dm$

از وضعیت l=n-dm و پارامترهای K و M عملکرد کاهش f_l 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 هزینه $\frac{2^l}{MX}$ در هر برخورد (که معادل $\frac{N}{M^{d+l}X}$ است) می‌توان پیدا کرد. در سطح یک، در کل $(2^{d+l}.M)$ 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 $2^{d-l}.\frac{N}{M^dX}$ 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 $2^{d-l}.\frac{N}{M^dX}$ زمان و $2^{d-l}.M$ حافظه دارد. زمان و حافظه پیچیدگی‌های سطوح باقی‌مانده دو به D همه M هستند، در نتیجه در مقایسه با تولید T_{pre} ناچیز هستند. پیچیدگی زمانی متعادل‌شده تولید جدول هلمن و سطح یک رابطه $T = (MX)^2 = 2^{d-l}.\frac{N}{M^{d-l}}$ است، که به $MX = 2^{d-l}.\frac{N}{M^dX}$ می‌تواند کاهش یابد. در جدول ۴، مبادله‌های قبلی ارائه‌شده 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- پیداکردن برخوردهای کامل

روش اساسی پیداکردن 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 پیداکردن برخوردها را تنها با $O(2^{n/2})$ محاسبه تابع چکیده‌ساز می‌دهد. به‌طور دقیق‌تر، تعداد محاسبه موردنیاز $\sqrt{\pi/2}\,2^{n/2}$. است.

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

روش		M	T	پارامترهای دیگر
K=4	برنشتین و همکاران (۱) $(T.M^2 = N)$	$2^{\frac{n}{4}}$	$2^{\frac{n}{2}}$	–
		$2^{\frac{n}{6}}$	$2^{\frac{2n}{3}}$	–
	الگوریتم پیشنهادشده	$2^{\frac{n}{4}}$	$2^{\frac{3n}{8}}$	$X=2^{\frac{n}{8}}$, $l=\frac{n}{2}$

[Downloaded from monadi.isc.org.ir on 2026-11-09]

	$(T.M^2=N)$	$2^{\frac{n}{6}}$	$2^{\frac{5n}{12}}$	$X=2^{\frac{n}{4}}$, $l=\frac{2n}{3}$
K=8	برنشتین و همکاران (۱) $(T.M^3=N)$	$2^{\frac{n}{5}}$ $2^{\frac{n}{6}}$	$2^{\frac{2n}{5}}$ $2^{\frac{n}{2}}$	– –
	برنشتین و همکاران (۳) $(T^2.M=N)$	$2^{\frac{n}{5}}$ $2^{\frac{n}{6}}$	$2^{\frac{2n}{5}}$ $2^{\frac{5n}{12}}$	– –
	الگوریتم پیشنهادشده $(T^2.M^2=N)$	$2^{\frac{n}{5}}$ $2^{\frac{n}{6}}$	$2^{\frac{3n}{10}}$ $2^{\frac{n}{3}}$	$X=2^{\frac{n}{10}}$, $l=\frac{2n}{5}$ $X=2^{\frac{n}{6}}$, $l=\frac{n}{2}$

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].

۱-۴- الگوریتم‌های کم‌حافظه[1]

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[2] با عنوان الگوریتم RHO برای فاکتورگیری و الگوریتم‌گسسته معرفی و بعد به جستجوی برخورد تعمیم داده شد. تابع چکیده‌ساز ابتدا از $\{0,1\}^*$ به $\{0,1\}^n$ محدود می‌شود، به‌طوری که $\{0,1\}^n\{0,1\}^n$ به $\{0,1\}^n$ 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 به‌دست 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 فلوید[3] [11]، الگوریتم برنت[4] [12]، با استفاده از تمایز نقاط [13] یا چند روش دیگر [14] می‌توان 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، به‌دلیل 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 [5] [15] مورد تجزیه و تحلیل قرار گرفت. گام اصلی الگوریتم محاسبه زنجیره‌ای از تکرار، با شروع از یک نقطه تصادفی و توقف زمانی‌که 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، به‌احتمال زیاد یک برخورد به‌دست 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 مشخص‌شده θ است.

۲-۴- پیداکردن تعداد کمی از برخورد به‌عنوان مثال «M i»

اگر به‌اندازهٔ کافی حافظه برای ذخیرهٔ 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 داده‌شده توسط ون اورشات C_{small} $\sqrt{2^n i}$. $\sqrt{\pi/2}$= $+2.5i/\theta$ است.

ویژگی‌های تمایز انتخاب شده است؛ پس درنهایت 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} = $\sqrt{\pi/2}$. $\sqrt{2^n i}/(1-2.5i/M)$ را نتیجه می‌دهد. اگر i≪ M باشد، C_{small} به صورت زیر می‌شود:

$$C_{small}=\sqrt{\pi/2}\,.\sqrt{2^n i}, \qquad (۹)$$

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

[1] Memory-less algorithms
[2] Pollard
[3] Floyd's algorithm
[4] Brent's algorithm
[5] Van Oorschot

[Downloaded from monadi.isc.org.ir on 2026-11-09]

۳-۴- پیداکردن تعداد زیادی از برخوردها به‌عنوان مثال «M i»

در این مورد، حافظه باید بازنویسی شود. آزمایش‌ها [15] نشان می‌دهند زمانی که حافظه پر است، پیچیدگی هر برخورد به‌طورتقریبی $2^n\theta\ /\ M+2/\theta$ است. این پیچیدگی به کمینه $\sqrt{8.2^n/M}$ برای $\sqrt{2M/2^n\theta}$= می‌رسد. به‌طور دقیق‌تر، ون اورشات آزمایش‌هایی برای تعیین ثابت‌های حقیقی انجام داده و پیچیدگی زیر حاصل شده است:

$$C= 5\sqrt{2^n/M}.i\ , \qquad (۱۰)$$

هنگامی که $2.25\sqrt{M/2^n\theta}$= است. یک عامل تسریع از $\sqrt{M}\ /\ 4$ نسبت به پیداکردن 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:

$$C\leq (\sqrt{\frac{\pi}{2}}+5\sqrt{\frac{i}{M}})\sqrt{2^n i}. \qquad (۱۱)$$

هنگامی که i «M یا M «i، یک دورهٔ 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- نزدیک‌برخوردها

w نزدیک‌برخورد یک جفت پیام x و x' ، به‌صورت $h(x)\oplus$ $h(x')\| \leq w$ است، که در آن $\|.\|$ وزن همینگ است. ابتدا برخی از نتایج در ارتباط با فاصله همینگ معرفی می‌شود.

تعریف ۱. اندازه یک توپ همینگ به شعاع w توسط $B_w(n)=\#\{x\in\{0,1\}^n: \|x\|\leq w\}$ نشان می‌دهند.

ویژگی۱. وجود دارد $B_w(n)=\sum_{i=0}^{w}\binom{n}{i}$

ویژگی۲. احتمال این که یک جفت x و x' تصادفی در w نزدیک‌برخورد نتیجه دهد برابر است با:

$$B_w(n)/2^n, \qquad (۱۲)$$

ویژگی۳. رابطه زیر برقرار است:

$$B_w(n)= B_w(n-1) + B_{w-1}(n-1), \qquad (۱۳)$$

قضیه ۱. نامساوی زیر نیز صدق می‌کند:

$$B_{w-1}(x) \leq \binom{x}{w}\frac{w}{x-2w+1}\ , \qquad (۱۴)$$

اثبات.

$$\begin{aligned}\frac{B_{w-1}(x)}{\binom{x}{w}} &= \frac{\binom{x}{w-1}+\binom{x}{w-2}+\binom{x}{w-3}+\cdots}{\binom{x}{w}}\\ &= \frac{w}{x-w+1}+\frac{w(w-1)}{(x-w+1)(x-w+2)}+\cdots\\ &\leq \frac{w}{x-w+1}+\left(\frac{w}{2-w+1}\right)^2+\cdots\\ &\leq \frac{\frac{w}{x-w+1}}{1-\frac{w}{x-w+1}} = \frac{w}{x-2w+1}\ ,\end{aligned} \qquad (۱۵)$$

(با استفاده از مجموع سری هندسی)

۵-۱- الگوریتم حافظه-کامل [1]

در این الگوریتم تعداد محاسبات موردنیاز برای پیداکردن یک نزدیک‌برخورد $i=\sqrt{\pi/2.2^n/B_w(n)}$ است. هم‌چنین حد پایین تعداد ارزیابی چکیده موردنیاز برای هر الگوریتم نزدیک‌برخورد دست‌کم $\sqrt{\pi/2.2^n/B_w(n)}$ محاسبه جهت دستیابی به w نزدیک‌برخورد با احتمال ناچیز است.

با این حال، این روش ساده نیاز به i. $B_w(n)$ = $\Omega(\sqrt{2^n.B_w(n)})$ حافظه قابل دسترس به جدول به اندازه $i = \Omega(\sqrt{2^n.B_w(n)})$ دارد. در مقابل حمله برخورد پیچیدگی را با استفاده از الگوریتم مرتب‌سازی، جدول چکیده و یا زنجیره تکرار نمی‌توان کاهش داد. درواقع در هر اجرای عملی ضعف حساب می‌شود [10].

۵-۲- استفاده از برخورد در یک تابع چکیده بریده‌شده

یک روش ساده، پیداکردن 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، زمانی‌که تابع چکیده‌ساز به $t =$

[1] Memory-full algorithm

[Downloaded from monadi.isc.org.ir on 2026-11-09]

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 - τ بیتی را پیدا و وزن همینگ τ بیت کوتاه‌شده را بررسی کرد و درنهایت 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

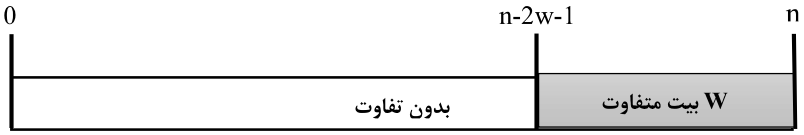
نخستین الگوریتم تعمیم ساده روش مبتنی بر کوتاه‌سازی در بخش ۵ است. مشاهده شدکه اگر τ بیت با $1 > 2w - \tau$ برش داده شود، احتمال این‌که یک برخورد در تابع کوتاه‌شده یک نزدیک‌برخورد در تابع چکیده‌ساز کامل باشد، به‌سرعت کاهش می‌یابد و نیاز به پیداکردن برخوردهای بسیاری دارد. در یک رویکرد حقیقی حافظه کم‌تر، پیداکردن i چنین برخورد نیاز به $i.\sqrt{\pi/2}.\sqrt{2^{n-\tau}}$. محاسبات دارد و برای به‌دست‌آوردن توسط روش برش‌دادن بیش از $2w$ - 1 بیت، کم‌تر وجود دارد. با این حال، با مقداری حافظه می‌تواند به‌طور قابل‌توجهی توسط عامل $\sqrt{i}$ در صورتی که $i \ll M$ یا $\sqrt{M}/4$ و $M \ll i$ (همان‌طور که در بخش۴ شرح داده شد) باشد، کاهش یابد.

درادامه، 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 به‌طو رقابل‌توجهی منجربه حمله‌های بهتر می‌تواند شود.

۶-۱- پیچیدگی

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

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



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

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

۶-۲- پیداکردن پارامترهای بهینه

در جهت پیداکردن خصوصیات جبری τ بهینه، تجزیه و تحلیل بخش ۴ را دنبال کرده و دو مورد برای پیچیدگی، بسته به رابطه بین i و M در نظر گرفتند.

با درنظرگرفتن تعداد کمی از بیت برش‌یافته و برخورد $2^{\tau}.B_w(\tau) \gg M$ پیچیدگی زیر به‌دست می‌آید:

[Downloaded from monadi.isc.org.ir on 2026-11-09]

$$=\sqrt{\pi/2}.\sqrt{2^{n-\tau}.2^{\tau}/B_w(\tau)}=C_{small}\ \sqrt{\pi/2}.\frac{2^{n/2}}{\sqrt{B_w(\tau)}}, \quad (۱۶)$$

وقتی که τ افزایش می‌یابد، این عبارت نیز افزایش پیدا می‌کند. با تعداد زیاد بیت برش‌یافته و تعداد زیاد برخوردها $2^{\tau}.B_w(\tau) \gg M$ پیچیدگی برابراست با:

$$C_{large}=\frac{5\sqrt{2^{n-\tau}/M}.2^{\tau}}{B_w(\tau)}=\frac{5.2^{n/2+\tau/2}}{B_w(\tau)\sqrt{M}}, \quad (۱۷)$$

با مطالعه تغییرات C_{large}:

$$(\tau-1)\leq C_{large}\,(\tau)C_{large} \quad (۱۸)$$

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

$$\frac{B_w(\tau)}{B_w(\tau-1)}\leq\sqrt{2}$$

$$\frac{B_w(\tau-1)+B_{w-1}(\tau-1)}{B_w(\tau-1)}\leq\sqrt{2}$$

$$\frac{B_{w-1}(\tau-1)}{B_w(\tau-1)}\leq\sqrt{2}-1$$

$$\frac{B_w(\tau-1)}{B_{w-1}(\tau-1)}\geq\sqrt{2}+1$$

$$\frac{\binom{\tau-1}{w}+B_{w-1}(\tau-1)}{B_{w-1}(\tau-1)}\geq\sqrt{2}+1$$

$$\frac{\binom{\tau-1}{w}}{B_{w-1}(\tau-1)}\geq\sqrt{2},$$

با استفاده از قضیه۱. $\binom{\tau-1}{w}/B_{w-1}(\tau-1)\geq$ $\frac{\tau-2w}{w}$. هنگامی‌که $\tau\geq(\sqrt{2}+2)w$، $\binom{\tau-1}{w}/B_{w-1}(\tau-1)\geq\sqrt{2}$ را نتیجه و $C(\tau)$ را افزایش می‌دهد. توجه داشته باشید که این فرمول تنها زمانی‌که $M\ll 2^{\tau}/B_w(\tau)$ یعنی برای مقادیر τ و این فرض که $\tau\geq(\sqrt{2}+2)w$است، صدق می‌کند. در این حوزه برای مقادیر مناسب پارامترها درست خواهد بود. به‌ویژه 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. زمانی‌که τ کوچک باشد، یعنی $M\ll 2^{\tau}/B_w(\tau)$پیچیدگی با τ کاهش می‌یابد؛ اما زمانی‌که τ بزرگ باشد، یعنی $M\ll 2^{\tau}/B_w(\tau)$با τ افزایش می‌یابد. با انتخاب بهینه τ، رابطه زیر ایجاد می‌شود:

$$M\approx 2^{\tau}/B_w(\tau) \quad (۱۹)$$

برای این اندازه τ، پیچیدگی به‌صورت زیر است:

$$C\approx 2^{n/2}/\sqrt{B_w(\tau)}, \quad (۲۰)$$

که این پیچیدگی بیشتر از پیچیدگی بهینه به‌دست آمده توسط الگوریتم حافظه‌کامل $2^{n/2}/\sqrt{B_w(n)}$ است؛ اما برای بیش‌تر پارامترها از حد $2^{n/2}/\sqrt{B_{w/2}(n)}$ (که الگوریتم مبتنی بر کد پوششی را محدود می‌کند) بهتر است.

τ بهینه درعمل. برای n و m داده شده، تخمین بهتری از τ بهینه می‌توان پیدا کرد. از حد بالای (۲۱) که حد بالای پیچیدگی زیر را می‌دهد، استفاده شده است:

$$C\leq C_{small}+C_{lg}= \left(\sqrt{\frac{\pi}{2}}+5\sqrt{\frac{2^{\tau}/B_w(\tau)}{M}}\right).\sqrt{\frac{2^n}{B_w(\tau)}}, \quad (۲۱)$$

برای پیداکردن یک بده‌بستان خوب، این حد را برای تمام مقادیرτ محاسبه و از مقداری τ که پایین‌ترین حد را می‌دهد، استفاده کردند. در مقاله [10] نمونه اعمال این حمله بر MD5 نشان داده است.

۷- نتایج

در این مقاله، حمله TMTO و روش‌های بهبود این حمله که تاکنون ارائه شده‌اند، معرفی و همچنین، الگوریتم عمومی جهت پیداکردن نزدیک‌برخوردها معرفی شد، که هردو الگوریتم پیشین مبتنی بر کوتاه‌سازی و مبتنی بر کدهای پوششی را تعمیم می‌دهد. علی‌رغم کارهای پیشین، هدف این الگوریتم مطالعه TMTO بوده است و درعمل قابل پیاده‌سازی است. لورنت ثابت کرد که با درنظرگرفتن یک مقدار حافظه عملی، با تغییردادن پارامترها بهبودی در عملکرد این الگوریتم می‌توان مشاهده کرد. این الگوریتم از الگوریتم جستجو موازی برخورد اورشات ایده گرفته است، که با حافظهٔ کم در زمان مناسبی کمتر از $i\sqrt{2^n}.i$ برخورد را می‌توان 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.

[Downloaded from monadi.isc.org.ir on 2026-11-09]

[2] Jean, J. and P.-A. Fouque. *Practical Near-Collisions and Collisions on Round-Reduced ECHO-256 ompression 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 Ihesis, 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. Addision 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.

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.

[Downloaded from monadi.isc.org.ir on 2026-11-09]

[Downloaded from monadi.isc.org.ir on 2026-11-09]