پنجمین کنفرانس انجمن رمز ایران
انجمن رمز ایران
Iranian Society of Cryptology
۲۴ الی ۲۶ مهر ماه ۱۳۸۶ دانشگاه علم و صنعت ایران

حمله‌ای کارآمد مبتنی بر ZDD به مولدهای رمز دنباله‌ای

محمد قاسم زاده
دانشگاه یزد – دانشکده کامپیوتر
m.ghasemzadeh@yazduni.ac.ir

کریستف واینل
دانشگاه پتسدام آلمان
meinel@hpi.uni-postdam.de

محمد حسین شاهزمانیان سیچانی
دانشگاه یزد – دانشکده ریاضی
shahzamanian@ stu.yazduni.ac.ir

مهسا شیر محمدی
دانشگاه یزد – دانشکده کامپیوتر
shirmohammadi@stu.yazduni.ac.ir

چکیده: *BDD* [1] ساختمان داده‌ای است که بطور کارآمدی در مباحث مختلف علوم و مهندسی کامپیوتر بکار گرفته شده است. این ساختمان داده در سال‌های اخیر در کنار روش حدس و تعیین، روش همبستگی و همچنین روش‌های جبری و آماری در تحلیل مولدهای رمز دنباله‌ای مطرح و بکار گرفته شده است. در واقع امروزه حمله مبتنی بر ساختمان داده *BDD*، یکی از موفق‌ترین و جدیدترین روش‌های ارائه شده در حمله به مولدهای رمز دنباله‌ای است.

در این مقاله ابتدا ساختار عمومی حمله مبتنی بر *BDD* را معرفی و به دنبال آن، حمله جدیدی به دسته‌ای از مولدهای رمز دنباله‌ای را پیشنهاد می کنیم. این حمله مبتنی بر گونه‌ای دیگر از *BDD* به نام *ZDD* [2] می باشد. در ادامه نشان می‌دهیم به چه ترتیب حمله پیشنهادی را می‌توان در خصوص مولد رمز دنباله‌ای E_0 که در مکانیزم امنیتی بلوتوث بکار برده می‌شود پیاده سازی نمود. در این رابطه الگوریتمی ارائه نموده و در محیط *C* با بسته نرم افزاری *CUDD* [3] تحت محیط *Linux* پیاده سازی و اجرا نموده‌ایم. نتایج آزمایش موید برتری حمله *ZDD* بر *BDD* می‌باشد. در این رابطه یک اثبات رسمی ریاضی نیز استخراج کرده‌ایم . این شواهد نشان می‌دهند که حد بالای پیچیدگی زمان محاسباتی حمله *ZDD* ، در حد بهترین زمان محاسباتی دیگر حملات پیشنهادی به مولد E_0 بوده و کاهشی به اندازه $O(2^8)$ در پیچیدگی حافظه محاسباتی بدنبال دارد.

واژه های کلیدی: تحلیل رمز، مولد رمز دنباله ای E_0 ، حمله *FBDD* ، ساختمان داده *ZDD*، نمودار تصمیم دودویی *BDD*

[1] Binary Decision Diagram
[2] Zero-suppressed Binary Decision Diagram
[3] Colorado University Decision Diagram

۱- مقدمه

در سیستم‌های رمزنگاری، دنباله‌های شبه تصادفی بوفور بکار گرفته می‌شوند. مولد دنباله‌های شبه تصادفی طبق اصول مطرح شده در *[1]* باید سه شرط: توزیع یکنواخت، استقلال و عدم همبستگی را برآورده کند. در پیاده‌سازی مولدهای رمز دنباله‌ای، بعلت برآورده شدن شروط مذکور و سادگی تحلیل جبری، عموماً از *LFSR* ها استفاده می‌کنند.

مولدهای رمز دنباله‌ای مبتنی بر *LFSR* شامل دو بخش خطی L و غیرخطی C می باشند. این مولدها را می‌توان بصورت $K=(L,C)$ نشان داد. این مولدها رشته بیت رمز Y را که از تابع $Y=C(L(k))$ بدست می‌آید، تولید می‌کنند. در اینجا k کلید رمز است. دنباله بیت رمز Y و دنباله متن ساده P، بصورت بیت به بیت با هم *XOR* می‌شوند تا دنباله رمز E را بوجود آورند: $e_i = p_i \oplus y_i$ *[2,3]*.

در حمله به این نوع مولدها با فرض دانستن سیستم رمز، هدف یافتن کلید k برای رشته بیت رمز است. در چنین سیستم‌هایی، k پیش از ارسال داده با کمک یکی از «پروتکل های مبادله کلید» بین فرستنده و گیرنده مبادله شده است.

BDD و گونه‌های دیگر آن مانند *ZDD* و *FBDD* از مدرن‌ترین ساختمان‌های داده در علوم کامپیوتر بحساب می‌آیند. این ساختمان داده‌ها نوعی گراف بدون دور و دارای راس می‌باشند، که می‌توانند فرمول‌های دودویی را بصورت فشرده و استاندارد نمایش دهند. از زمان ابداع این ساختمان‌های داده تاکنون کارهای تحقیقاتی موفق بسیاری جهت بکارگیری آنها در مسائل علمی و کاربردی صورت گرفته است *[4]*.

حمله به مولدهای رمز دنباله‌ای پیشینه‌ای طولانی دارد و تاکنون حملات زیادی به این نوع مولدها ارائه شده است. در سال‌های اخیر حمله‌ای متفاوت توسط *Krause* در *[2]*، به مولدهای رمز دنباله ای مبتنی بر *LFSR* پیشنهاد شده است. این حمله از ساختمان داده *FBDD* استفاده می‌کند. او نشان داده است که این مولدها در برابر حمله پیشنهادی آسیب پذیر می‌باشند. این حمله تاکنون یکی از بهترین حملات به مولدهای رمز دنباله‌ای بوده است. بطور مشابه *[5]*، با استفاده از همان ایده و کاهش آن به کمک *OBDD* حمله‌ای را بر روی مولد E_0 طراحی کرده است. در ادامه بررسی این حملات *Krause* در *[6]*، آن دسته از مولدهایی که می‌توانند از حمله *OBDD* استفاده کنند را بررسی کرده و الگوریتم مربوطه را باز هم کاهش داده است. در این مقاله، روشی جدید در حمله به مولدهای رمز دنباله‌ای معرفی می‌کنیم. در این حمله که برای اولین بار از ساختمان داده *ZDD* در تحلیل رمز استفاده کرده است، با کاهش پیچیدگی قابل ملاحظه‌ای روبرو می‌شویم و صحت این کاهش را به کمک اثبات ریاضی و نتایج آزمایش نشان می‌دهیم.

این مقاله بصورت زیر سازماندهی شده است. ابتدا در بخش دوم به تعاریف مقدماتی می‌پردازیم. در این بخش مولد E_0 را معرفی می‌کنیم. در ادامه به معرفی ساختمان داده‌های *BDD* و *ZDD* می‌پردازیم. بخش سوم به حمله پیشنهادی می‌پردازد. بهمین منظور ابتدا الگوریتم حمله عمومی *FBDD* را معرفی و بکارگیری آن را توسط *OBDD* در مورد مولد رمز دنباله‌ای E_0 بررسی می‌کنیم و سپس حمله *ZDD* را پیشنهاد می‌دهیم و در بخش چهارم به محاسبه پیچیدگی الگوریتم پیشنهادی توسط اثباتی ریاضی می‌پردازیم.

۲- تعاریف مقدماتی

۲-۱- مولد رمز دنباله‌ای E_0

E_0 نوعی مولد رمز دنباله‌ای مبتنی بر *LFSR* است. مولدهای رمز دنباله‌ای مبتنی بر *LFSR*، دسته ای از مولدهایند که از دو بخش خطی و غیرخطی تشکیل شده اند. مولد E_0 با کلید ۱۳۲ بیتی k، مقداردهی اولیه می شود، که ۱۲۸ بیت آن برای مقداردهی اولیه بخش خطی و ۴ بیت آن برای مقداردهی بخش غیرخطی است. مولد پس از مقداردهی اولیه، با گذر از بخش خطی L رشته بیت Z را به عنوان ورودی بخش غیرخطی C تولید می کند و خروجی بخش غیرخطی، رشته بیت رمز Y است؛ $Y=C(L(k))$. بخش خطی L در مولد E_0، از چهار *LFSR* تشکیل شده است. طول $|L_0|=25$،

$|L_1|=31$، $|L_2|=33$ و $|L_3|=39$ و توابع آن ها عبارتند از:

$$p_0(x)=x^{25}+x^{20}+x^{12}+x^8+1$$
$$p_1(x)=x^{31}+x^{24}+x^{16}+x^{12}+1$$
$$p_2(x)=x^{33}+x^{28}+x^{24}+x^4+1$$
$$p_3(x)=x^{39}+x^{36}+x^{28}+x^4+1$$

خروجی بخش خطی L مجموع چهار بیت خروجی L_i هاست. بخش غیرخطی E_0 را می توان بصورت یک ماشین متناهی حالت[4] $C_{E_0}:(Q,\Sigma,\Gamma,I,F,\delta)$ در نظر گرفت، که در آن، مجموعه حالات ماشین $Q=\{q_i:0\le i\le 16\}$ ، مجموعه ورودی ماشین $\Sigma=\{0,1,2,3,4\}$ و مجموعه خروجی ماشین $\Gamma=\{0,1\}$ می باشند، I نیز بیانگر حالات آغازین ماشین می باشد که بوسیله ۴ بیت از ۱۳۲ بیت مقدار دهی اولیه شده است و F نیز مجموعه حالات پایانی است که در اینجا مجموعه تهی است[5,6,7].

مجموعه قوانین ماشین $\delta\subseteq Q\times\Sigma\times\Gamma\times Q$ که دارای اعضایی به صورت (q_n,a,b,q_{n+1}) می‌باشد بیانگر توابع انتقال ماشین $(q_n,a)\to q_{n+1}$ و توابع خروجی ماشین $(q_n,a)\to b$ است. مجموعه قوانین ماشین در جدول ۱ نشان داده شده است.

جدول ۱: مجموعه قوانین ماشین متناهی حالت در مولد E_0

	a=0		a=1		a=2		a=3		a=4	
q_n	b	q_{n+1}	b	q_{n+1}	b	q_{n+1}	b	q_{n+1}	b	q_{n+1}
0	0	0	1	0	0	4	1	4	0	8
1	0	12	1	12	0	8	1	8	0	4
2	0	4	1	4	0	0	1	0	0	12
3	0	8	1	8	0	12	1	12	0	0
4	1	5	0	1	1	1	0	13	1	13
5	1	9	0	13	1	13	0	1	1	1
6	1	1	0	5	1	5	0	9	1	9
7	1	13	0	9	1	9	0	5	1	5
8	0	14	1	14	0	2	1	2	0	6
9	0	2	1	2	0	14	1	14	0	10
10	0	10	1	10	0	6	1	6	0	2
11	0	6	1	6	0	10	1	10	0	14
12	1	11	0	7	1	7	0	3	1	3
13	1	7	0	11	1	11	0	15	1	15
14	1	15	0	3	1	3	0	7	1	7
15	1	3	0	15	1	15	0	11	1	11

۲-۲- معرفی *ROBDD* و *ZDD*

درخت دودویی تصمیم[5]، یک گراف جهت دار و بدون دور[6] است، که بر روی مجموعه‌ای از متغیرهای بولی $\forall m\in N, X_m=\{x_1,\dots,x_m\}$ تعریف می‌شود؛ چنین درختی نمایشگر تابع دودویی بر روی متغیرهای X_m است. یک درخت دودویی تصمیم *BDT* شامل دو نوع گره است. گره از نوع برگ که گره پایانی T نیز خوانده می‌شود و گره‌های داخلی N که با یکی از متغیرهای $x_i\in X_m$ برچسب *var(N)* خورده‌اند. هر گره داخلی دارای دو فرزند می باشد، *low(N)* برای $x_i=0$ و *high(N)* برای $x_i=1$.

یک گره منحصر بفرد به نام N_0، با درجه ورود ۰ ریشه *BDT* را تشکیل می‌دهد. کلیه گره های داخلی دارای درجه ورود ۱ می‌باشند. هر گره پایانی T ، دارای برچسب ۰ یا ۱ است و مطابق با برچسب آن *0-T* یا *1-T* نامیده می شود. هر تابع دودویی مبتنی بر m متغیر، 2^m ترکیب برای ورودی دارد. هر انتساب $b\in\{0,1\}^m$ را می‌توان به وسیله مسیری که از گره ریشه N_0 شروع و به یک گره پایانی خاتمه می‌یابد، نمایش داد. مقدار تابع به ازای انتساب b، که بصورت *F(b)* نیز نامبرده می‌شود برابر با مقدار گره پایانی است.

نمودار دودویی تصمیم *BDD* یک نوع گراف کارآمد می‌باشد که از روی *BDT* یک تابع دودویی بدست می‌آید. یک *BDD* دارای دو گره پایانی با برچسب ۰ و ۱ می‌باشد. زیرگراف های مشابه در *BDT* ، حین تبدیل به *BDD* درهم ادغام می‌شوند. دو زیرگراف G_1 و G_2، مشابه اتلاق می شوند اگر در ریشه دو گراف g_1 و g_2 داشته باشیم: $\text{var}(g_1)=\text{var}(g_2)$، $high(g_1)=high(g_2)$ و $low(g_1)=low(g_2)$.

همچنین گره‌های بی‌اثر در *BDT* نیز حین تبدیل به *BDD* حذف می‌شوند. گره‌ای مانند N بی‌اثر خوانده می‌شود که $low(N)=high(N)$ یعنی صرفنظر از این که مقدار N چه باشد به گره یکسانی می‌رسیم.

FBDD[7] نوعی *BDD* است که در همه ی مسیرها از گره‌ی ریشه N_0 به یکی از گره‌های T، هر متغیر بولی x_i حداکثر یکبار دیده شود.

گراف پیشگوی G[8]، نوعی *FBDD* بر روی X_m است که فقط یک گره پایانی T دارد و در هر مسیر بر روی G همه‌ی m متغیر X_m، یکبار دیده می‌شوند. گراف پیشگو به منظور

[4] Finite State Automata
[5] Binary Decision Tree
[6] DAG (Directed Acyclic Graph)
[7] Free Binary Decision Diagram
[8] Oracle Graph

محاسبه توابع بولی بکار نمی‌رود، بلکه برای نمایش مجموعه‌ای از ترتیب های ممکن و معتبر ایفای نقش می کند.

G-FBDD نوعی *FBDD* است که در هر مسیر P_F در آن، ترتیب رخداد متغیرها متناظر با یکی از مسیرهای P_G ازگراف پیشگو باشد.

ROBDD [9] نوعی *G-FBDD* است که گراف پیشگوی G در آن تنها شامل یک مسیر باشد؛ به بیان دیگر بجای گراف پیشگو یک بردار داریم که فقط یک ترتیب برای ملاقات متغیرها در کلیه مسیرهای *ROBDD* را نشان می‌دهد.

ROBDD یا به عبارتی*OBDD*، دارای این خاصیت مهم می‌باشد که یک «نمایش رسمی»[10] در اختیار می‌گذارد؛ به بیان دیگر اگر چند عبارت مختلف برای یک تابع دودویی داشته باشیم با در نظر گرفتن بردار ترتیب یکسان، *ROBDD* همه‌ی آنها دقیقاً یکسان است. از آنجایی که کلیه موارد قابل حذف و ادغام در ساخت آن لحاظ می‌شود، *ROBDD* نمایش فشرده‌ای از توابع بولی در اختیار می‌گذارد، در حالیکه روش‌های دیگر نمایش توابع دودویی مانند جدول درستی، *BDT* و جدول کارنو نیاز به حافظه ای از مرتبه نمایی دارند.

ZDD گونه‌ای دیگر از *BDD* است، که آن هم به نوبه خود از کاهش *BDT* بدست می‌آید. یک *ZDD* نیز، فقط دارای دو گره پایانی *0-T* و *1-T* است. زیر گراف های مشابه در *BDT*، حین تبدیل به *ZDD* نیز ادغام می شوند، در حالیکه در *ZDD* گره ای که *high(N)* آن به *0-T* متصل باشد، حذف می شود.

اساساً *ZDD* برای نمایش و انجام عملیات کارآمد بر روی «مجموعه‌های خلوت» ابداع شده است. لذا در حالیکه *ROBDD* در رابطه با توابع دودویی بهتر عمل می کند، *ZDD* در رابطه با مسائل درگیر با مجموعه‌ها خصوصاً مجموعه‌های خلوت بهتر عمل می‌کند.

می‌توان اثبات کرد که اندازه *ZDD* هیچ‌گاه بیش از تعداد عناصر مجموعه‌ی مورد نظر نمی‌باشد، در حالیکه در رابطه با *ROBDD* حد بالا تعداد زیرمجموعه‌ها ضربدر تعداد اقلامی است که در هر یک می توانند بیایند*[2,4,8]*.

[9] Reduced Ordered Binary Decision Diagram
[10] Canonical representation

۳- حمله *ZDD* به مولد رمز دنباله‌ای E_0

۳-۱- حمله عمومی *FBDD* به مولدهای رمز دنباله‌ای

در الگوریتم عمومی که *[2]* برای حمله به مولدهای رمز دنباله‌ای پیشنهاد داده است؛ ابتدا به کاهش مساله می‌پردازد. در حمله به مولدهای رمز دنباله‌ای، رشته بیت رمز Y معلوم است و هدف از حمله محاسبه کلید k است. از آن جا که در یک *LFSR* بیت‌های اولیه تولید شده در خروجی، مقدار اولیه *LFSR* است، بنابراین $Z = L(k)$ در بیت های اولیه خود مقدار k را در بردارد. پس می‌توان حمله به چنین سیستم هایی را، به یافتن Z ای کاهش داد؛ بصورتی که:

۱. رشته بیتی قابل تولید توسط بخش خطی مولد L باشد

۲. $C(Z)$ پیشوند رشته رمز معلوم Y باشد.

برای $m \geq 1$ و رشته بیت $Z = \{0,1\}^m$ تعریف می‌کنیم :

- G_m^c گراف پیشگویی است که ترتیب خوانده شدن بیت‌های رشته ی Z را بوسیله بخش غیرخطی C نشان می‌دهد.
- R_m ، یک گراف $G_m^c - FBDD$ است که در مورد قابل تولید بودن Z توسط L ، تصمیم می‌گیرد.
- Q_m ، یک گراف $G_m^c - FBDD$ است که در مورد پیشوند Y بودن $C(Z)$، تصمیم می‌گیرد.
- P_m ، یک گراف $G_m^c - FBDD$ است که در مورد قابل تولید بودن Z توسط L و پیشوند y بودن $C(Z)$، تصمیم می‌گیرد.

در *[2]* با فرض n بیتی بودن کلید k ، مقدار m^* را، که تعداد بیت‌های متوالی لازم برای یافتن کلید k است، را محاسبه کرده است.

بنابراین طبق الگوریتم زیر k قابل محاسبه است:

۱. $Q_n \rightarrow P$.

۲. برای m از $n+1$ تا m^* تکرار کن:

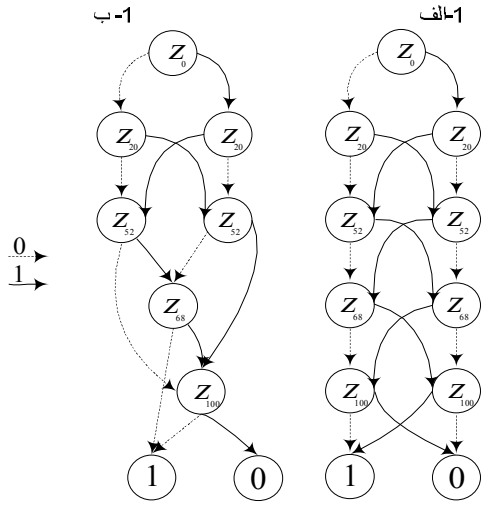
$$(P \wedge Q_m \wedge R_m) \rightarrow P$$

۳. مقدار z^* را که $P(z^*) = 1$ را برگردان.

به بیان دیگر حلقه آنقدر تکرار می شود که P_{m^*} فقط یک انتساب $z^* \in \{0,1\}^{m^*}$ برای $P(z^*) = 1$ داشته باشد.

۳-۲- کاهش حمله عمومی *FBDD* با ارائه حمله *OBDD* به مولد رمز دنباله E_0

مرجع *[5]* با ارائه حمله‌ی *OBDD* به مولد رمز دنباله‌ای E_0، حمله عمومی *FBDD* را در مورد این مولد کاهش داد، و در مرجع *[6]* حمله *OBDD* به دسته‌ای از مولدهای رمز دنباله‌ای تعمیم داده شد.



شکل ۱- الف گراف z_{100} بوسیله *BDD* و ۱- ب گراف z_{100} بوسیله *ZDD* است.

در حمله *OBDD* رشته بیت های خروجی $L(k)$ به صورت

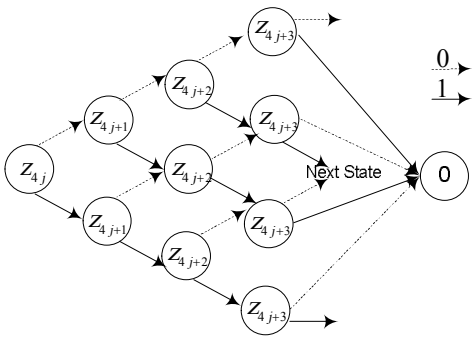
$$Z=(\ldots,z_{4j},z_{4j+1},z_{4j+2},z_{4j+3},\ldots)$$

نشان گذاری شده است که دسته بیت $z_{4j+i}\in L_i(k_i)$. بنابراین با اعمال این ترتیب، معادلات زیر برای بخش خطی L بدست می آید.

$$\forall i=4j: z_i=z_{i-32}\oplus z_{i-48}\oplus z_{i-80}\oplus z_{i-100}$$
$$\forall i=4j+1: z_i=z_{i-48}\oplus z_{i-64}\oplus z_{i-96}\oplus z_{i-124}$$
$$\forall i=4j+2: z_i=z_{i-16}\oplus z_{i-96}\oplus z_{i-112}\oplus z_{i-132}$$
$$\forall i=4j+3: z_i=z_{i-16}\oplus z_{i-112}\oplus z_{i-144}\oplus z_{i-156}$$

سپس بر اساس معادلات بدست آمده ، با ساخت *OBDD* هر z_i، گراف R_m را تشکیل داده است. شکل ۱- الف ، *OBDD* ساخته شده برای محاسبه z_{100} را نشان می‌دهد. در ساخت *OBDD* رشته بیت های هر L_i، $|L_i|$ بیت اول فراخوانی می‌شوند، هدف الگوریتم نیز محاسبه ی $|L_i|$ بیت اول هر *LFSR* است، بنابراین با توجه به معادلات نیاز به ساخت *OBDD* برای $z_j:|L|\le j\le 4|L|$ است. تعداد *OBDD* های ساخته شده در این بخش $3\times|L|=3\times128=384$ است *[5]*.

هر *OBDD* دارای ۵ متغیر و ۱۱ گره است، بنابراین ۴۲۲۴ گره مورد نیاز است. در محاسبه‌ی گراف Q_m از ساختاری زنجیره‌ای شکل۲، که زنجیره جمع چهار بیت است، استفاده شده است. در این ساختار از بلوک‌های ۱۶ زنجیره‌ای معادل با ۱۶ حالت ماشین متناهی حالت استفاده شده است، که هرمسیر در هر زنجیره‌ی یک بلوک، با قیاس بیت معلوم در رشته بیت رمز Y با بیت خروجی ماشین متناهی حالت، یا به *0-T* منجر می شود یا به حالت مناسبی در بلوک بعدی متصل می شود. مطابق با شکل۲، هر زنجیره از ۴ متغیر و ۱۰ گره تشکیل شده است بنابراین در دنباله ۱۲۸ بلوکی، $10\times16\times128=20,480$ گره مورد نیاز است، البته *[5]* بیان کرده است که به علت قوانین کاهش در *OBDD*، این تعداد به حدود $14,500$ گره تقلیل می‌یابد.



شکل۲: زنجیره جمع چهاربیتی

۳-۳- حمله *ZDD* به مولد رمز دنباله E_0

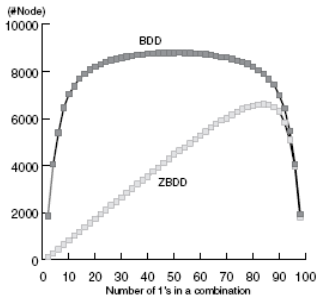
پیش از بیان حمله *ZDD* ، ابتدا چگونگی عملکرد بهتر *ZDD* نسبت به *OBDD* در حمله را، شرح می‌دهیم. *OBDD* به منظور نمایش و انجام عملیات بر روی توابع دودویی، و *ZDD* به منظور نمایش و انجام عملیات بر روی مجموعه‌ی ترکیب‌ها *[9]* ابداع شده‌اند، بنابراین *ZDD* در این‌گونه مسائل کارآمدتر از *OBDD* است.

ترکیب n آیتم، را می‌توان به صورت یک بردار n بیتی $(x_1,\ldots,x_n)$ نمایش داد که $x_i\in\{0,1\}$ بیانگر حضور آیتم iام در ترکیب مربوطه است. بنابراین مجموعه ترکیب‌ها را می‌توان به یک تابع دودویی با n بیت ورودی متناظر کرد. در تابع دودویی مذکور، هدف مشخص کردن ترکیب‌های ممکن در مجموعه جواب مسئله است، که هر ترکیب بواسطه حضور

یا عدم حضور آیتم‌ها بصورت منحصر بفردی تعریف می‌شوند. این چنین توابعی را تابع مشخصه مجموعه می‌گویند. عملیات اجتماع، اشتراک و تفاضل می‌توانند به این مجموع‌ها اعمال شوند.

OBDD در نمایش توابع مشخصه مجموعه‌های ترکیبی، بصورت فشرده نسبت به ساختمان داده‌های دیگر بهتر عمل می کند، علی رغم این بهبود، بعلت قانون نامناسب *OBDD* در برخورد با مسائل ترکیبی، مسئله‌ای جدی بوجود می‌آید. *ZDD* در واقع *OBDD* اصلاح شده، با تعویض قانون حذف است که منجر به نمایش بهتر در مسائل ترکیبی می‌شود*[8]*.

درمحاسبه‌ی ترکیب‌های مجموعه نمایش داده شده در *ZDD* ، تمامی مسیرهای ممکن به *1-T* را بررسی می‌کنیم در هر مسیر ملاقات گره X_i به معنای حضور متغیر X_i در ترکیب، و عدم ملاقات گره X_i به معنای عدم حضور متغیر X_i در ترکیب است *[9]*.



شکل۳: نمودار عملکرد BDD و ZDD در مسائل ترکیبی خلوت

در شکل۳، نموداری برای نمایش عملکرد *ZDD* و*BDD* در مسائل ترکیبی رسم شده است. نمودار مذکور نشان می‌دهد که در نمایش توابع ترکیبی *ZDD* بهتر از *BDD* عمل می‌کند بخصوص اگر ترکیب، خلوت باشد به بیان دیگر تعداد ۱ ها در آن کم باشد*[8]*.

در تحلیل رمز مولدهای دنباله‌ای، هدف بررسی حالات ممکن کلید است تا کلید متناظر با دنباله رمز معلوم بدست آید.حمله *FBDD* به دسته‌ای از مولدهای رمز دنباله‌ای را، می‌توان بوسیله *OBDD* کاهش داد. این دسته از مولدها ترتیب یکسان ورودی در ساخت گراف‌های Q_m, R_m و در نتیجه S_m را برآورده می‌کنند*[5,6]*. در این دسته از مولدهای رمز دنباله‌ای می‌توان بخش غیرخطی را بصورت ماشین متناهی حالتی نظیر کرد. حمله‌ی *ZDD* در مورد همین دسته از مولدهای رمز دنباله‌ای مطرح می‌شود.

حمله‌ی *ZDD* را با پیاده سازی در مورد مولد رمز دنباله‌ای E_0 تشریح می‌کنیم، روش مذکور برای همه‌ی مولدهای این دسته قابل تعمیم است.

در حمله پیشنهادی مبتنی بر *ZDD* به مولد رمز دنباله‌ای E_0 ، از الگوریتم کلی حمله *FBDD* استفاده می‌شود و تفاوت در نحوه پیاده سازی گراف‌های مورد نظر و بهینه سازی این پیاده سازی است.

در حمله‌ی *ZDD* به مولدهای رمز دنباله‌ای E_0 ، در پیاده سازی گراف R_m ، از ایده *[5]* استفاده شده است، و تفاوت در پیاده سازی ساخت z_j بوسیله ساختمان داده *ZDD* بجای *OBDD* است. شکل ۱- ب *ZDD* ساخته شده برای محاسبه z_{100} نشان می دهد. مطابق با شکل هر *ZDD* ساخته شده دارای *4* متغیر و ۹ گره می باشد، بنابراین در پیاده سازی *ZDD* ۳۸۴ لازم، ۳۴۵۶ گره لازم است.

در محاسبه‌ی گراف Q_m در حمله *ZDD* ، ایده زیر را پیشنهاد می‌دهیم. از آنجا که ماشین متناهی حالت C_{E_0} دارای شانزده حالت است برای نشان‌گذاری هر حالت ماشین، از چهار متغیر $0 \le i \le 3, q_n^i$ استفاده می‌کنیم.بنابراین می‌توان تابع زیر را محاسبه کرد:

Q_m تابع $Q_m = F(q_{m+1}^3, q_{m+1}^2, q_{m+1}^1, z_{4m+3}, z_{4m+2}, z_{4m+1}, \ldots z_0)$ تابعی متشکل از $4m+4$ متغیر است که بیانگر تمامی مسیرهای ممکن در ماشین متناهی حالت بعد از m مرحله ورود و خروج است. تابع Q_m بصورت زیر ساخته می‌شود.

۱. اگر قانون انتقال $(q_m, a) \to q_{m+1}$ و متناظر با آن قانون خروجی $(q_m, a) \to b$ را در ماشین متناهی حالت C_{E_0} داشته باشیم، آنگاه در صورتی که $b_m = b$:

۱،۱. معادل متناظر q_m و q_{m+1} را بر اساس q_m^i بدست می‌آوریم.

$$q_m = q_m^{0^*} \wedge q_m^{1^*} \wedge q_m^{2^*} \wedge q_m^{3^*}$$

که $q_m^{i^*}$ برابر با q_m^i یا متمم آن $q_m^{i'}$ است.

۲،۱. حاصلضرب زیر را به ازای $\sum z_{4m+i} = a$ بدست می‌آوریم:

$$X_j = (q_{m+1}^{3\,*} \wedge q_{m+1}^{2\,*} \wedge q_{m+1}^{1\,*} \wedge q_{m+1}^{0\,*} \wedge z_{4m+3} \wedge z_{4m+2} \wedge z_{4m+1} \wedge z_{4m} \wedge q_m^{3\,*} \wedge q_m^{2\,*} \wedge q_m^{1\,*} \wedge q_m^{0\,*})$$

۲. حال با بدست آوردن حاصلضرب متناظر با همه قوانین به محاسبه Q_m می‌پردازیم:

$$Q_m = ((X_0 \wedge Q_{m-1}) \vee \ldots \vee (X_j \wedge Q_{m-1}))$$

درمحاسبه تابع Q_0 ، تابع Q_{m-1} را با *1* بولی مقداردهی می‌کنیم.

۳. در عبارات بدست آمده Q ، متغیرهای $q_m^{i\,*}$ را حذف می‌کنیم، تابع حاصل Q_m است.

توجه داریم که در نهایت برای حل مساله به Q_{128} نیاز داریم. گراف Q_m ساخته شده به خوبی در مورد پیشوند Y بودن $C(z)$ تصمیم می‌گیرد، و با بررسی تمامی انتساب‌های منتهی به *1-T* می‌توان Z هایی را که منجر به پیشوند مشترک با Y می شود، بدست آورد.

حمله پیشنهادی *ZDD* ، بر این اساس ساخته شده است که در هر مرحله گذر از ماشین متناهی حالت آنچه که مهم است و می‌بایست در حافظه ماشین بماند، حالت انتهایی ماشین و رشته بیتی است که در رسیدن به آن حالت توسط ماشین متناهی حالت خوانده شده است.

گراف Q_m محاسبه شده، خلوت است و این به علت الزامات رمزهای شبه تصادفی است. رمزهای شبه تصادفی باید شرط توزیع یکنواخت را برآورده کنند. به بیان دیگر احتمال اتفاق ۰ و ۱، یکسان است ، بهمین دلیل گراف Q_m ، خلوت می‌باشد.

در پیاده سازی الگوریتم پیشنهادی بوسیله *ZDD* ، مساله را بصورت مجموعه‌ای از ترکیب‌ها بررسی می‌کنیم. در هر بار محاسبه Q_m در واقع مجموعه‌ای از ترکیب‌های ممکن ورودی و حالت پایانی ماشین با ورود آن ترکیب را خواهیم داشت.

در ساختمان داده *ZDD* ، برای عملیات بر روی مجموعه ترکیب‌ها توابع بسیاری ساخته شده‌است از جمله توابع مجموعه‌ای مانند اجتماع، اشتراک ، تفاضل و توابعی مانند :

- *Z.Oneset(N)* : مجموعه‌ای از ترکیب‌های ممکن را در مجموعه مرجع که متغیر N در آنها حضور دارد، با حذف N بر می‌گردانند.
- *Z.Count(N)* : تعداد ترکیب ها در Z.

در الگوریتم مذکور، محاسبه Q_m طبق شبه کد زیر قابل محاسبه است :

```
for ∀ element ∈ δ
  if ((q_m, a) → q_{m+1} ∧ (q_m, a) → b ∧ (b = b_m))
        { q_m* ← ZDDIntersect(q_m^{3*}, q_m^{2*}, q_m^{1*}, q_m^{0*})
        q_{m+1}* ← ZDDIntersect(q_{m+1}^{3*}, q_{m+1}^{2*}, q_{m+1}^{1*}, q_{m+1}^{0*})
        for ∀ z_{4m+i}, 0 ≤ i ≤ 3
        {
        if Σ z_{4m+i} = a
        X_j = ZDDIntersect(q_{m+1}*, z_{4m+3}, z_{4m+2}, z_{4m+1}
        , z_{4m}, q_m*)
}}
Q ← ZDDUnion(∀j, ZDDIntersect(X_j,
Q_{m-1}.Oneset(q_m*), if((q_m*)_{X_j} == (q_m*)_{Q_{m-1}})
For every q_m*
  Q_m ← Q.Onset(q_m*)
```

٤- تحلیل پیچیدگی حمله *ZDD* به مولد E_0

در تحلیل پیچیدگی ابتدا توابع بکار رفته را بطور کلی بیان می‌کنیم.

- ساخت یک *ZDD* برای تابع $F(x_0,\ldots.x_{i-1})$ پیچیدگی زمانی برابر $O(|G_F|)$ دارد. $|G_F|$ تعداد گره‌های گراف G_F در ساخت تابع F را نشان می دهد.
- ساخت *ZDD* اشتراک و اجتماع بین دو گراف F و G، پیچیدگی برابر $O(|G_F||G_G|)$ دارد.

با بررسی پیچیدگی توابع بالا، مشاهده می شود که پیچیدگی زمان عملیاتی به سایز *ZDD* ساخته شده در هر مرحله، بستگی دارد. در الگوریتم ارائه شده در هر مرحله، چهار متغیر به ازای بیت‌های خروجی از بخش غیرخطی و چهار متغیر به ازای حالات انتهایی ماشین جایگزین چهار متغیر دیگر می شود. از طرفی در چهار متغیر ورودی به ماشین متناهی حالت، با در نظر گرفتن شرط $\Sigma z_{4m+i} = a$ و نحوه ساخت z_{4m+i} ، انتخابی آزادنه وجود ندارد. در طی $|L_0|$ مرحله اول، خروجی بخش خطی کاملا مقادیر اولیه کلید است و شرطی بواسطه بخش خطی اعمال نمی‌شود، تنها شرط مجموع مقید چهار متغیر به a است که بهمین علت سه متغیر انتخاب آزادنه دارند و تعداد حالات انتخابشان 2^3 خواهد بود، در طی مراحل $|L_0|$ تا $|L_1|$ ، خروجی L_0، وابسته به مقادیر قبلی است،

وخروجی سه *LFSR* دیگر همچنان مقادیر اولیه کلید است بنابراین در شرط مجموع مقید چهار متغیر به a انتخابی آزادنه برای Z_m وجود ندارد که بهمین علت دو متغیر انتخاب آزادنه دارند و تعداد حالات انتخابشان 2^2 خواهد بود، بهمین ترتیب در طی مراحل $|L_1|$ تا $|L_2|$ یک متغیر انتخاب آزادنه دارد و تعداد حالات انتخابش 2^1 خواهد بود، و در $|L_2|$ تا $|L_3|$ انتخابی آزادنه وجود ندارد، و در مراحل بعد از $|L_3|$ در هر مرحله بعلت b_m متناظر تعداد حالات انتخابی نصف می‌شود. از طرف دیگر در هر مسیر گراف P_m تعداد متغیرهای ظاهر شده حداکثر $4m$ است، اما در پیاده‌سازی مطرح شده بوسیله *ZDD* و با توجه به ورودی‌های ماشین متناهی متوسط متغیرهای ظاهر شده برابر است با :

$$\frac{\binom{4}{1}\times m+\binom{4}{2}\times 2m+\binom{4}{3}\times 3m+\binom{4}{4}\times 4m}{\binom{4}{0}+\binom{4}{1}+\binom{4}{2}+\binom{4}{3}+\binom{4}{4}}=2m$$

بنابراین در هر مسیر بطور متوسط به این تعداد متغیر ظاهر می‌شود پس با توجه به دو بحث انجام شده داریم :

$$\begin{cases} m\le|L_0|:|P_m|=2m\times 2^{3m} \\ |L_0|\le m\le|L_1|:|P_m|=2m\times 2^{3|L_0|}\times 2^{2m-|L_0|} \\ |L_1|\le m\le|L_2|:|P_m|=2m\times 2^{3|L_0|}\times 2^{2(|L_1|-|L_0|)}\times 2^{m-|L1|} \\ |L_2|\le m\le|L_{31}|:|P_m|=2m\times 2^{3|L_0|}\times 2^{2(|L_1|-|L_0|)}\times 2^{|L_2|-|L_1|} \\ |L_3|\le m:|P_m|=2m\times 2^{3|L_0|}\times 2^{2(|L_1|-|L_0|)}\times 2^{|L_2|-|L_1|}\times 2^{|L3|-m} \end{cases}$$

از طرف دیگر با استناد بر الگوریتم اشتراک بین دو گراف، حد دیگری برای گراف بدست می‌آید. گراف $|P_m|$ در هر مرحله با اشتراک بر روی R_{4m} و Q_m بدست می‌آید. گراف R در بیت‌های اولیه هر*LFSR* گرافی تک گره‌ای است ودر $3|L_i|$ بیت بعدی طبق شکل۱- الف ساخته می‌شود. $|Q_m|$ در پیاده سازی عملی حدوداً دارای سایز 2^{14} است، بنابراین:

$$\begin{cases} m\le|L_0|:|P_m|=|Q_m|\times|R_m|=|Q_m| \\ |L_0|\le m\le|L_1|:|P_m|=|Q_m|\times 2^{m-|L_0|-1} \\ |L_1|\le m\le|L_2|:|P_m|=|Q_m|\times 2^{m-|L_0|-1}\times 2^{m-|L_1|-1} \\ |L_2|\le m\le|L_{31}|:|P_m|=|Q_m|\times 2^{m-|L_0|-1}\times 2^{m-|L_1|-1}\times 2^{m-|L_2|-1} \\ |L_3|\le m:|P_m|=|Q_m|\times 2^{m-|L_0|-1}\times 2^{m-|L_1|-1}\times 2^{m-|L_2|-1}\times 2^{m-|L_{3-1}|-1} \end{cases}$$

با اشتراک حدود بدست آمده، حد بالای پیچیدگی تئوری در مسئله 2^{82} است؛ که البته بسیار بزرگتر از حد بدست آمده در عمل خواهد بود. حد تئوری در حمله *ZDD* از حد تئوری بدست آمده از حمله *OBDD* به اندازه $O(2^8)$ کاهش نشان داده است.

۵- نتیجه‌گیری

در این تحقیق نشان داده‌ایم که به چه ترتیب می‌توان با اتکا به یک ساختمان داده به نام *ZDD* به بهبود حمله به مولدهای رمز دنباله‌ای دست یافت. در واقع بخش مهمی از برتری روش پیشنهادی نشأت گرفته از قابلیت‌های ویژه *ZDD* در نمایش فشرده اطلاعات و عملگرهای کارآمد تعریف شده، روی آن است. در حمله پیشنهادی به کاهش قابل ملاحظه‌ای در پیچیدگی محاسباتی نسبت به حمله *FBDD* دست یافته‌ایم. به طوری که یک اثبات ریاضی نشان می دهد که کاهشی به اندازه $O(2^8)$ نسبت به حمله *FBDD* حاصل شده است.

۶ - مراجع

[1] M.J.B. Robshaw. Stream Ciphers. RSA Laboratories Technical Report TR-701Version 2.0|July 25, 1995.
[2] M. Krause. BDD-Based Cryptanalysis of Keystream Generators. Proceedings of the EUROCRYPT 2002,LNCS 2332, pp. 222–237.
[3] D. Stegemann. BDD-based Cryptanalysis of the A5/1 Keystream Generator - Experimental Results., SASC - The State of the Art of Stream Ciphers, October 14-15, 2004, Bruges, Belgium.
[4] M. Ghasemzade. An efficient data structure in Computer Science, 12th International CSI Computer Conference (CSICC'2007), Tehran, Iran, February 2007.
[5] Y. Shaked and A. Wool. Cryptanalysis of the Bluetooth E0 cipher using OBDD's. In Proc. 9th Information Security Conference, LNCS 4176,pages 187-202, Samos, Greece, August 2006.
[6] M. Krause. OBDD-Based Cryptanalysis of Oblivious Keystream Generators. Theor. Comp. Sys
[7] S.R. Fluhrer and S.Lucks. Analysis of the E0 encryption system. In Proc. 8th Workshop on Selected Areas in Cryptography, LNCS 2259. Springer-Verlag, 2001.
[8] S. Minato: "Zero-Suppressed BDDs and Their Applications", International Journal on Software Tools for Technology Transfer, Vol. 3, No. 2, pp. 156-170, Springer, May 2001
[9] A. Mishchenko, "An introduction to zero-suppressed BDD", Technical report, Portland State University, June 2001.
[10] M. Krause and D. Stegemann. Reducing the space complexity of BDD-based attacks on keystream generators. In 13th annual Fast Software Encryption Workshop 2006.
[11] F. Somenzi. CUDD Package, Release 2.3.1. http://vlsi.Colorado.EDU/~fabio/CUDD/cuddIntro.html