A Cryptanalytic Time-Memory Trade-Off

Thomas Glue Rasmussen
Thomas Nørbo Jensen

11. maj 2012

Indhold

1 Introduktion

Mange problemer introducerer muligheden for at lave en trade-off mellem tid(t) og hukommelse(m) i stedet for kun strengt enten at bruge tid eller hukommelse. Dette er også tilfældet i forbindelse med kryptoanalyse, hvor der ønskes at finde frem til en nøgle ud fra viden omkring plaintext og ciphertext. Ved brute force metoden bruges tid $T = N$ og ved tabelopslag bruges hukommelse $M = N$. Ved benyttelse af trade-off mellem tid og hukommelse er det som beskrevet i [1] muligt at bruge $M = m \cdot t$ hukommelse og $T = t^2$ tid under antagelse af at $m \cdot t^2 = N$. Ved at sætte $m = t = N^{1/3}$ har vi at $M = T = N^{2/3}$, hvilket er bedre end de 2 andre nævnte algoritmer.
Et af kravene til denne metode er at vælge et stykke plaintext P på forhånd for at kunne opbygge tabellen. Derefter er det nødvendigt at få brugeren til at kode P ved hjælp af brugerens valgte nøgle K. Dette betragtes som et chosen plaintext angreb, hvilket kan virke som en urealistisk form for angreb, men det er muligt i stedet for at lave en tabel med et P som ud fra f.eks. statistisk analyse er meget brugt og derefter lede efter blokke i ciphertext som optræder ofte. Derefter benyttes algoritmen på disse blokke for at finde en mulig nøgle, som kan bruges til at dekryptere resten med. Hvis det er muligt at dekryptere teksten korrekt er den rigtige nøgle fundet.
Metoden, der bliver beskrevet, danner også fundament for "Rainbow Tables", hvilke er beskrevet i [2]. Disse kan forholdsvis nemt findes på internettet og bruges i praksis til forskellige angreb. Eksempelvis bliver der i rapporten beskrevet et angreb på "MS Windows", der udnytter "Lan Manager". Egentligt er angrebet forholdsvis nemt at forsvare sig imod, men af forskellige årsager har forskellige systemer ikke valgt at implementere forsvaret. I "Lan Manager" er årsagen eksempelvis bagudkompatibilitet med ældre versioner.

2 Problemstilling

I dette afsnit beskrives, hvilke forhold vi arbejder under.

I dette tilfælde benytter vi DES, men denne måde at angribe problemet på kan bruges til at invertere alle envejs funktioner. Da vi er interesseret i at finde frem til nøglen K og vi arbejder med chosen plaintext P_0, fastholder vi P_0 og varierer K, hvor ved det man normalt forbinder ved kryptering vil K fastholdes og P_0 varieres.
Denne problemstilling er ikke helt irrelevant, da det er den typiske måde et operativsystem gemmer kodeord på. (Fastholder P_0, lader K være kodeordet og genererer C_0, der bruges til at sammenligne med)
Overordnet set er ideen at bruge kryptosystemet til at definere en funktion f.

Udover kryptosystemet indeholder f også en reduktionsfunktion R. Denne reduktionsfunktion bruges til at reducere antallet af bits i C_0 således, at det svarer til antallet af bits i K. Derved er det muligt at benytte f til at mappe fra nøglerummet over i sig selv. Da vi her benytter DES betyder det at vi reducerer fra 64 bits til 56 bits. Denne reduktion kan være simpel som for eksempel at fjerne de første 8 bits.

2.1 Definitioner

I løbet af rapporten bruges følgende

- N angiver antallet af nøgler
- $\hat{N} = \{1, \ldots N\}$ angiver nøglerummet, hvilket er mængden af alle mulige nøgler
- P_0 er den kendte plaintext
- $S_K()$ angiver krypteringssystemet med K som nøgle
- C_0 angiver den krypterede tekst, så $C_0 = S_K(P_0)$
- $R : 2^{64} \rightarrow 2^{56}$ angiver en reduktionensfunktion fra 64 til 56 bit. Dette kunne eksempelvis være at ignorere de sidste 8 bit.
- $f(K) = R(S_K(P_0)),\ K \in \hat{N}$ er en funktion, der mapper nøglerummet over i sig selv.

3 Metoden

I dette afsnit gennemgås og analyseres metoden. Overordnet set er ideen at konstruere en opslagstabel, der bruger $O(m)$ hukommelse. Denne tabel kan så bruges til at forsøge på at gætte nøglen i tid $O(t)$. Da metoden kun dækker en del af nøglerummet vil sandsynligheden for at gætte nøglen afhænge af størrelserne af m og t, men det kan ikke betale sig at øge dem alt for meget, hvilket bliver gennemgået i analyse afsnittet.

3.1 Opbygning af tabellen

Metoden bruger en opslagstabel som datastruktur. Konstruktionen af denne beskrives i dette afsnit. Det skal bemærkes at tabellen kan genbruges til lignende angreb, så den skal kun konstrueres en gang for at kunne angribe et specifikt system flere gange.

Til algoritmen benyttes en opslagstabel, som bliver genereret på forhånd. Dette gøres ved at vælge m uniformt tilfældige startpunkter, der gemmes i opslagstabellen. På det første startpunkt $SP_1 = x_{1,0}$ anvendes funktionen f

til at generere $x_{1,1}$. Dette fortsættes t gange indtil $x_{1,t} = EP_1$ er beregnet. Man kan tænke på det som en kæde med t elementer, men kun SP_1 og EP_1 gemmes i tabellen for at overholde hukommelseskravet på $O(m)$. Dette gøres for alle m startpunkter, så tabellen til sidst indeholder m start- samt slutpunkter. Det antages at det er muligt at foretage et opslag i tabellen i $O(1)$ tid, hvilket eksempelvis ville være muligt med et "hash-map". På grund af den måde funktionen f mapper nøgler på er der risiko for overlap i kæderne. Det er nemt at overbevise sig selv om at hvis 2 kæder først er kollideret vil de være sammenflettet i resten af deres forløb. Overlappene skyldes blandt andet funktionen R, der definerer f og betyder at 2 forskellige nøgler kan mappes til den samme nøgle. Sådanne overlap betyder at nøglerummet ikke dækkes optimalt ud fra de valgte startpunkter. Desuden er der også risiko for kredse i en kæde da f kan mappe til alle værdier, også nøgler som allerede optræder i kæden. Som ved overlap vil en kreds også reducere dækningen af nøglerummet.

3.2 Brug af tabellen

På nuværende tidspunkt er der blevet kontrueret en række kæder, der har til formål at dække en del af nøglerummet. For ikke at bruge for meget hukommelse er kun start og slutpunkterne af kæderne blevet gemt i en tabel, så spørgsmålet er nu, hvordan skal tabellen bruges?
Vi er givet $C_0 = S_K(P_0)$ og skal på en eller anden måde bruge tabellen til at gætte K. Den nemmeste måde at overbevise sig selv om metoden er nok at se første par iterationer.
Det første der gøres er at sætte $Y_1 = R(C_0)$. Vi kan da bemærke at $Y_1 = R(C_0) = R(S_K(P_o)) = f(K)$ jævnfør definitionen af f. Herefter foretages der et enkelt opslag i tabellen for at se om Y_1 optræder som et af endepunkterne. Hvis det er tilfældet at $Y_1 = EP_i = x_{i,t}$ er påstanden at $x_{i,t-1}$ er et muligt bud på nøglen K, hvilket følger af at $R(S_K(P_o)) = Y_1 = EP_i = x_{i,t} = R(S_{x,t-1}(P_o))$.
Det er dog ikke garanteret at $x_{i,t-1} = K$, da flere kæder godt vil kunne have EP_i som endepunkt. Det kan være at K ligger som andet sidste element i en kæde med EP_i som slutpunkt, der ville være opstået, hvis et andet startpunkt var blevet valgt. Dette kaldes en falsk alarm.
For at finde $x_{i,t-1}$ skal der bruges $O(t)$ beregninger, da hele kæden skal genereres igen fra startpunktet. Herefter kan det nemt testes om $x_{i,t-1} = K$ ved at se om $C_0 = S_{x_{i,t-1}}(P_0)$.
Hvis Y_1 derimod ikke optræder som et endepunkt i tabellen beregnes $Y_2 = f(Y_1) = f^2(K)$ og der checkes for om Y_2 optræder blandt endepunkterne i tabellen. Er det tilfældet at $Y_2 = EP_j = x_{j,t}$ er påstanden at $x_{j,t-2}$ er et muligt bud på nøglen K. Dette kan ses da $f^2(K) = Y_2 = EP_j = x_{j,t} = f^2(x_{j,t-2})$. Igen kan vi være uheldige, at det bare er en falsk alarm.
Så hvis Y_{i-1} ikke optræder som et endepunkt sættes $Y_i = f(Y_{i-1}) = f^i(K)$.

Hvis Y_i optræder som et endepunkt altså $Y_i = EP_j = x_{j,t}$ vil $x_{j,t-i}$ være et muligt K. Dette er tilfædet da $f^i(K) = Y_i = EP_j = x_{j,t} = f^i(x_{j,t-i})$. Dette trin gentages t gange.
Da det antages et opslag i tabellen tager $O(1)$ tid og hvis det endvidere antages, at der kun er et konstant antal falske alarmer vil metoden tage $O(t)$ tid.

3.3 Analyse

Da metoden ikke nødvendigvis dækker hele nøglerummet er et åbenlyst spørgsmål, hvor stor sandsynligheden $P(S)$ er for at gætte nøglen. Det første vi kan bemærke er at $P(S) \leq \frac{m \cdot t}{N}$, da vi dækker $m \cdot t$ ud af N mulige nøgler. Grunden til at dette er en øvre grænse er at vi kan risikere at kæderne overlapper. Intuitivt giver det god mening at jo større m og t bliver jo værre vil overlappet også være, så spørgsmålet er, hvor store kan vi vælge m og t og stadig vinde noget på det?

Sætning 1 *Hvis $f : \hat{N} \to \hat{N}$ er en stokastisk funktion og nøglen K er valgt uniformt i $\hat{N}$, så er $P(S)$ begrænset af*

$$P(S) \geq \frac{1}{N} \cdot \sum_{i=1}^{m} \sum_{j=0}^{t-1} (\frac{N - i \cdot t}{N})^{j+1} \tag{1}$$

Bevis Lader $A \subseteq \hat{N}$ være mængden af nøgler, der af dækket af kæderne. Det vil da gælde at

$$P(S) = \frac{E[|A|]}{N} \tag{2}$$

Ved at definere den stokastiske variable $X_{i,j}$, der er 1, hvis nøglen på plads j i kæde nr i ($k_{i,j}$) ikke er set før og 0 ellers, fås følgende relation

$$\begin{aligned} E[|A|] &= E[\sum_{i=1}^{m} \sum_{j=0}^{t-1} X_{i,j}] \\ &= \sum_{i=1}^{m} \sum_{j=0}^{t-1} E[X_{i,j}] \\ &= \sum_{i=1}^{m} \sum_{j=0}^{t-1} Pr(k_{i,j} \text{ } er \text{ } ny) \end{aligned}$$

$Pr(k_{i,j}\ er\ ny)$ angiver sandsynligheden for at $k_{i,j}$ ikke er set før. Da nøglen $k_{i,j}$ kun har mulighed for at være ny hvis $k_{i,j-1}$ er ny fås

$$\begin{aligned} Pr(k_{i,j}\ er\ ny) &\geq Pr(k_{i,0}, k_{i,1}, \ldots, k_{i,j}\ er\ alle\ nye) \\ &= Pr(k_{i,0}\ er\ ny) \cdot Pr(k_{i,1}\ er\ ny | k_{i,0}\ er\ ny) \ldots \\ &Pr(k_{i,j}\ er\ ny | k_{i,0}, \ldots k_{i,j-1}\ er\ nye) \\ &= \frac{N - |A_{i,0}|}{N} \frac{N - |A_{i,0}| - 1}{N} \cdots \frac{N - |A_{i,0}| - j}{N} \end{aligned}$$

Da der maksimalt er t forskellige elementer i hver række vil hver faktor i ovenstående være større end $(\frac{N-i\cdot t}{N})^{j+1}$. Dette fører til

$$\begin{aligned} Pr(k_{i,j}\ er\ ny) &\geq (\frac{N - i \cdot t}{N})^{j+1} \\ &\Downarrow \\ P(S) &\geq \frac{1}{N} \cdot \sum_{i=1}^{m} \sum_{j=0}^{t-1} (\frac{N - i \cdot t}{N})^{j+1} \end{aligned}$$

Ifølge sætning 1 kan det ikke betale sig at øge m og t særlig meget over det punkt, hvor $m \cdot t^2 = N$. Dette følger da $(\frac{N-i\cdot t}{N})^{j+1} \approx e^{-ijt/N}$. Hvis $m \cdot t^2 >> N$ vil $(\frac{N-i\cdot t}{N})^{j+1} \approx e^{-ijt/N} \approx 0$, så vil leddene i summen, hvor $i \cdot j^2 >> N$ ikke bidrage med specielt meget.
Kigger vi på det andet tilfælde, hvor $m \cdot t^2 << N$ vil $(\frac{N-i\cdot t}{N})^{j+1} \approx e^{-ijt/N} \approx 1$, så vil den samlede sandsynlighed for at gætte nøglen være cirka $\frac{m\cdot t}{N}$.
Hvis m og t begge vælges så de er store og $m \cdot t^2 = N$ kan 1 evalueres numerisk til cirka at være $\frac{0{,}8\cdot m\cdot t}{N}$. Det vil da gælde at $\frac{0{,}8\cdot m\cdot t}{N} \leq P(S) \leq \frac{m\cdot t}{N}$.
Sætning 1 viser ligeledes at for typiske valg af m og t vil sandsynligheden for at gætte nøglen være forholdsvis lille. Eksempelvis

$$\begin{aligned} m = t &= N^{\frac{1}{3}} \\ &\Downarrow \\ P(S) &\approx \frac{1}{N^{\frac{1}{3}}} \end{aligned}$$

Dette kan løses ved at generere flere forskellige tabeller, hvor reduktionsfunktionen R varieres. Hvis nøglen ikke findes i den første tabel prøves den næste osv.
Strukturen af de forskellige tabeller vil variere meget da ved forskellige valg af R vil det at det samme punkt optræder i 2 forskellige tabeller ikke betyde at den samme kæde optræder i de 2 tabeller.
Et andet problem er at vi ikke nødvendigvis er garanteret køretiden på $O(t)$, da der kan være mange falske alarmer, hvilket kan påvirke køretiden

Sætning 2 *Det forventede antal falske alarmer pr tabel $E(F)$ er begrænset af*

$$E(F) \leq \frac{m \cdot t \cdot (t+1)}{2 \cdot N} \tag{3}$$

Jævnfør sætning 2 vil valget af m og t så $m \cdot t^2 \approx N >> 1$ give et forventet antal falske alarmer pr tabel på $\frac{1}{2}$. Da hver falsk alarm koster $O(t)$ operationer vil den samlede køretid forventet set stadig være $O(t)$.
Hvis vi kigger lidt videre på det tidligere eksempel, hvor $m = t = N^{1/3}$, vil der jf. sætning 2 forventet være cirka $\frac{1}{2}$ falsk alarm pr. tabel, hvilket øger beregningstiden med cirka 50 procent. Så beregningstiden vil stadig være $O(t)$ på trods af de falske alarmer.

3.4 Forsvar imod angrebet

Hele metoden afhænger af at P_0 er kendt, så selv små ændringer i P_0 vil gøre tabellen ubrugelig. En metode til at foretage ændringer i P_0 er "saltning". Ideen er at generere et tilfældigt bitmønster kaldet $SALT$ og kryptere på følgende måde $S_K(P_0 \oplus SALT)$. $SALT$ må gerne være offentligt tilgængeligt, da vi bliver nødt til at generere tabeller for alle værdier af $SALT$ for at bruge metoden, hvilket for tilpas store saltværdier gør at hukommelsesforbruget i praksis vil blive for stor.

4 Rainbow Tables

Metoden, der er blevet beskrevet danner fundament for "Rainbow Tables", hvilke er beskrevet i [2]. "Rainbow Tables" har et par fordele i forhold til denne metode eksempelvis

- Kæderne vil ikke lave lige så meget overlap
- Der skal bruges færre forskellige tabeller for at gætte en nøgle, hvilket forbedrer køretiden

Overordnet set er ideen at variere reduktionsfunktionen når det næste led i kæden skal genereres. Dette mindsker overlappet da kæder ikke nødvendigvis fletter sammen hvis de rammer hinanden. Det gode ved dette er blandt andet at der opnås en bedre nedre grænse for sandsynligheden for at gætte nøglen. I artiklen gennemgås der ligeledes et praktisk resultat, der er opnået med "Rainbow Tables", hvor de bryder "MS Windows" kodeord inden for cirka 14 sekunder med en succesrate på 99,9 procent. Dette er opnået med et hukommelsesforbrug på 1,4 gigabyte.

5 Konklusion

Algoritmen af Martin E. Hellman giver et godt indblik i hvordan det er muligt at benytte trade-off teknikker til at forbedre kryptoanalytiske køretider. Alt efter systemet som algoritmen skal afvikles på, er det muligt at begrænse hukommelsforbruget eller tidsforbruget således at det tilpasses systemets krav.

Algoritmen har en god sandsynlighed for succes, og dette kan nemt skaleres ved hjælp af antallet af tabeller i brug. Da forskellen i tabeller kun er reduktionsfunktion, er det hukommelsen som sætter begrænsningen for antal. Desuden er det nemt at dele tabeller ud til forskellige processorer og derved parallelisere processen.

Til sidst skal nævnes at algoritmen nemt kan forsvares imod ved hjælp af værktøjer såsom salting og cipher-block chaining, men dette underbygger også vigtigheden i at bruge de nævnte værktøjer for at undgå svagheder i de oprindelige krypteringsmetoder.

Litteratur

[1] Martin E. Hellman. A cryptanalytic time - memory trade-off. *IEEE*, 1980.

[2] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. I *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, side 617-630, 2003.