ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta jaderná a fyzikálně inženýrská

Katedra matematiky

BAKALÁŘSKÁ PRÁCE

Hašovací funkce a kombinatorika na slovech

Hash Functions and Combinatorics on Words

Vypracoval: Jan Legerský

Školitel: Ing. Ľubomíra Balková, Ph.D.

Akademický rok: 2012/2013

Čestné prohlášení

Prohlašuji na tomto místě, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze dne 25. června 2013

Jan Legerský

Poděkování

Děkuji Ing. Ľubomíře Balkové, Ph.D., za věnovaný čas, vstřícný přístup a celkově vynikající vedení mé bakalářské práce. Dále bych chtěl poděkovat všem členům TIGRa a prof. Juhovi Kortelainenovi za podnětné diskuze. Velký dík patří také mým rodičům za to, že mě vždy podporovali ve studiu.

Jan Legerský

Název práce: **Hašovací funkce a kombinatorika na slovech**

Autor: Jan Legerský

Obor: Inženýrská informatika

Zaměření: Softwarové inženýrství a matematická informatika

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Ľubomíra Balková, Ph.D., KM FJFI, ČVUT v Praze

Konzultant: —

Abstrakt: Tato práce pojednává o hašovacích funkcích, zejména pak o jejich odolnosti proti hledání druhého vzoru. Shrnuje známé útoky a popisuje obranu proti nim v podobě ditherování, kterou navrhl R. Rivest. Ke zkoumání kvality ditherovaných hašovacích funkcí jsou využity poznatky z kombinatoriky na slovech. Jsou navržena vhodná ditherační slova a popsáno jejich generování. K implementaci příkladu ditherované funkce je použita kompresní funkce MD5. Dále je popsán útok ukazující nevhodnost určité konstrukce ditherované kompresní funkce. Nakonec je předveden útok na zvláštní kompresní funkci, která zakomponovává do každé iterace i předchozí bloky zprávy.

Klíčová slova: Hašovací funkce, kombinatorika na slovech, ditherování.

Title: **Hash Functions and Combinatorics on Words**

Author: Jan Legerský

Abstract: The thesis deals with hash functions, especially with their second preimage resistance. The already known attacks are summarized and dithering, a proposal of R. Rivest, is described as a protection against them. The knowledge of combinatorics on words is used to analyze the quality of dithered hash functions. Some suitable dither words are proposed with description of their generating. An example of dithered hash function is implemented using the compression function MD5. The impropriety of a specific construction of dithered compression function is shown by description of an attack. Also an attack on a special compression function that uses previous blocks in each iteration is described.

Key words: Hash Function, Combinatorics on Words, Dithering.

Obsah

Seznam symbolů

Symbol	Popis
$\mathbb{N}$	množina přirozených čísel $\{1, 2, 3, \dots\}$
$\widehat{n}$	množina čísel $\{1, 2, 3, \dots, n\}$
$\lfloor n \rfloor$	dolní celá část čísla n
$\#L$	mohutnost množiny L
H	množina haší délky n, tzn. $\{0,1\}^n$
B	množina bloků zpráv délky m, tzn. $\{0,1\}^m$
f	hašovací funkce
F	kompresní funkce
$\mathcal{O}$	asymptotická složitost
$\|\|$	řetězení bitových řetězců, zpráv
$\oplus$	bitové xor $(1 \oplus 1 = 0, 1 \oplus 0 = 0 \oplus 1 = 0, 0 \oplus 0 = 0)$
$\mathcal{A}$	abeceda
$\mathcal{A}^*$	monoid konečných slov nad $\mathcal{A}$
ϵ	prázdné slovo nad $\mathcal{A}$
w	konečné slovo nad $\mathcal{A}$
$\|w\|$	délka slova w
$\mathbf{d}$	nekonečné slovo nad $\mathcal{A}$
$\mathcal{L}(\mathbf{d})$	jazyk slova $\mathbf{d}$, tzn. množina všech faktorů slova $\mathbf{d}$
$\mathcal{L}_m(\mathbf{d})$	množina všech faktorů slova $\mathbf{d}$ délky m
$\mathcal{C}_\mathbf{d}(n)$	faktorová komplexita slova $\mathbf{d}$

Úvod

Tato práce se zabývá kryptografickými hašovacími funkcemi. Ukázalo se, že nejčastěji používaná iterativní konstrukce navržená Merklem a Damgårdem má jisté slabiny. Objevilo se množství útoků, které ukazují, že používaná konstrukce není tak silná, jak se původně myslelo. Jako obranu proti útoku na druhý vzor navrhl R. Rivest metodu ditherování, kterou se zde budeme zabývat. Při zkoumání ditherovaných hašovacích funkcí využijeme také některé poznatky z kombinatoriky na slovech. V závěru se budeme věnovat také jiným konstrukcím než ditherování.

V kapitole 1 popíšeme základní vlastnosti hašovacích funkcí a jejich konstrukci a naznačíme jejich široké využití. Dále shrneme v kapitole 2 několik výsledků z diskrétní pravděpodobnosti, teorie náhodných grafů a kryptografie. Následně tyto výsledky použijeme v kapitolách 3 a 4 pro popis útoků na kolize, multikolize a hledání druhého vzoru.

V kapitole 5 uvedeme základní pojmy a poznatky z kombinatoriky na slovech, které využijeme ke konstrukci slov s určitými vlastnostmi. Zmíníme také algoritmus, který se používá pro generovaní prefixů nekonečných slov. S takto připravenými pojmy můžeme poté v kapitole 6 zavést ditherování hašovací funkce vhodným nekonečným slovem. Popíšeme útoky na ditherované hašovací funkce a u jednoho z nich, kite generátoru, opravíme složitost offline fáze uváděnou autory. V závěru kapitoly ukážeme, která ze zkonstruovaných slov umí známým útokům zabránit.

V kapitole 7 je krátce popsána implementace hašovací funkce ditherované zkonstruovanými slovy, která využívá kompresní funkce MD5. Také předvedeme útok na určitý nevhodný způsob využití klasické kompresní funkce pro implementaci ditherované.

Na závěr se v kapitole 8 budeme věnovat dalším možným iterativním konstrukcím, a sice wide-pipe a HAIFA, které byly využity v soutěži NIST o nový standard SHA-3. Jako příklad ne příliš vhodné iterativní konstrukce uvedeme LAB mód, který počítá v každé iteraci i s předcházejícími bloky. Navrhneme analogii Jouxova útoku, čímž dokážeme, že zabudování předchozích bloků do kompresní funkce nepřináší žádné významné zlepšení odolnosti vůči hledání multikolizí.

Kapitola 1

Hašovací funkce

Hašovací funkce jsou důležité kryptografické nástroje, jejichž hlavním úkolem je přiřazovat zprávě libovolné délky otisk (hash) pevně dané délky. Známým požadavkem je, aby hašování bylo jednocestné, tady aby zahašování zprávy proběhlo rychle, ale aby bylo výpočetně nedosažitelné najít k zadanému otisku zpět zprávu. V této kapitole popíšeme, jaké další požadavky na vlastnosti hašovacích funkcí klademe, jak se konstruují, a zmíníme jejich použití.

1.1 Vlastnosti

Označme $\mathsf{H} = \{0,1\}^n$ množinu všech možných haší délky n. Hašovací funkce $f : \{0,1\}^N \to \mathsf{H}$ je zobrazení přiřazující zprávě M libovolné délky N otisk pevně dané délky n, která má následující vlastnosti [16]:

- Odolnost proti nalezení vzoru – k dané haši h_{target} je výpočetně nemožné nalézt zprávu M takovou, že $f(M_{\text{target}}) = h_{\text{target}}$.
- Odolnost proti kolizím – je výpočetně nemožné nalézt různé zprávy M a M' takové, že $f(M) = f(M')$.
- Odolnost proti nalezení druhého vzoru – k dané M_{target} je výpočetně nemožné nalézt zprávu M takovou, že $f(M) = f(M_{\text{target}})$.
- Snadný výpočet $f(M)$ – v čase $\mathcal{O}(N)$ a konstantní paměti.

Možných zpráv je mnohem více než možných haší, proto funkce nemůže být prostá a velké množství zpráv se zobrazí na stejnou haš. Přesto požadujeme, aby jejich nalezení bylo výpočetně nedosažitelné, čímž máme na mysli, že současné počítače neumí takovou úlohu řešit v reálném čase. To znamená co nejvíce se přiblížit teoretickým hodnotám, tedy složitostem útoků proti náhodnému orákulu (zobrazení produkující obrazy s rovnoměrným rozdělením), a zvolit n takové, že množství operací daného útoku není proveditelné.

Složitosti útoků hrubou silou, neboli útoků na náhodné orákulum, jsou následující:

- Hledání kolizí – řádově $2^{\frac{n}{2}}$ volání, odpovídá narozeninovému paradoxu, viz vztah (2.1).
- Hledání vzoru – řádově 2^n volání, viz vztah (2.2).
- Hledání druhého vzoru – řádově 2^n volání, analogické hledání vzoru.

Poznamenejme, že výše uvedené vlastnosti nejsou zcela nezávislé – umíme-li hledat druhý vzor, umíme nalézt i kolizi. Stačí ke zprávě M nalézt druhý vzor M' a máme kolizi $f(M) = f(M')$. Přesto požadujeme od hašovací funkce splnění všech požadavků, neboť je podstatné, s jakou složitostí umíme daný útok provést. Zajímavé je, že umíme-li nalézt vzor, neznamená to automaticky, že umíme najít druhý vzor, protože může k dané haši existovat pouze jeden.

1.2 Konstrukce

Nejčastější konstrukcí je iterativní Merkleovo-Damgårdovo paradigma založené na použití kompresní funkce F. Nese název po tvůrcích, kteří je v roce 1989 navrhli nezávisle na sobě na stejné konferenci CRYPTO '89 [7, 17] .

Haš $f(M)$ zprávy M délky menší než $N = 2^t - 1$ spočteme následovně (viz obrázek 1.1):

1. Rozdělíme zprávu M na bloky M_k velikosti m:

$$M_1 M_2 \dots M_{\ell-1} \underbrace{M'_\ell||1||00\dots0||\text{length}(M)}_{m \text{ bitů}} .$$

   Zbytek zprávy M'_ℓ je doplněn na délku m bitů jedničkou a nulami tak, aby posledních t bitů bylo vyhrazeno na binární zápis délky zprávy. Takové úpravě se říká Merkleovo-Damgårdovo zesílení. Množinu všech možných bloků označme $\mathsf{B} = \{0,1\}^m$.

2. Iterujeme ℓ-krát kompresní funkci $F : \mathsf{H} \times \mathsf{B} \to \mathsf{H}$ a vyrábíme tak průběžné haše (kontexty) h_i:

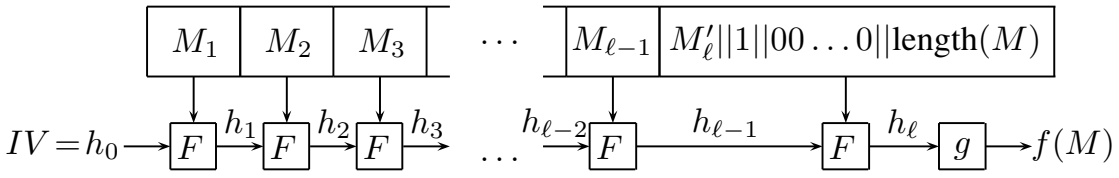
$$\begin{aligned} h_0 &= IV\,, \\ h_i &= F(h_{i-1}, M_i)\,, \end{aligned}$$

   přičemž h_0 je nějaká pevně daná inicializační hodnota IV.

3. Získáme otisk:

$$f(M) = g(h_\ell)\,.$$

   Za funkci g se často volí identita.



Obrázek 1.1: Merkleova-Damgårdova konstrukce hašovací funkce.

Touto konstrukcí převedli autoři zkoumání odolnosti proti nalezení kolizí celé hašovací funkce na zkoumání odolnosti pouze kompresní funkce. To shrnuje následující věta:

Věta 1 (Damgårdova-Merkleova)**.** *Bezkoliznost kompresní funkce implikuje při použití Merkleova-Damgårdova zesílení bezkoliznost celé hašovací funkce.*

Důkaz 1. *Postupujme sporem. Předpokládejme, že existují dvě různé zprávy M a M' délek ℓ a ℓ' bloků takové, že $f(M) = f(M')$. Pak buď $M_\ell \neq M'_{\ell'}$, což je spor s bezkolizností kompresní funkce, nebo $M_\ell = M'_{\ell'}$ a tudíž i $\ell = \ell'$ díky Merkleovu-Damgårdovu zesílení. V tom případě zjistíme, jestli jsou shodné předcházející bloky – jestliže ne, máme spor s bezkolizností kompresní funkce, pokud ano, postoupíme na předešlé bloky. Tak buď dojdeme v některém kroku do sporu s bezkolizností kompresní funkce, nebo $M = M'$, což je spor s předpokladem existence kolize.*

Stačí tedy zkoumat bezkoliznost kompresní funkce, což je mnohem snazší, protože transformuje vstup délky pouze $m+n$ na výstup délky n oproti celé hašovací funkci, u které může být vstup libovolně dlouhý.

Známým příkladem kompresní funkce je Daviesova-Meyerova konstrukce využívající blokové šifry $E(k, x)$, která je vůči klíči jednocestná. Konstrukce je následující:

$$F(h_{i-1}, M_i) = E(M_i, h_{i-1}) \oplus h_{i-1}\,.$$

Jako klíč se tedy použije blok zprávy M_i a pomocí něj se zašifruje průběžná haš h_{i-1}. Následně se ještě přixoruje haš h_{i-1}. Toto schéma využívají i velmi známé funkce jako MD5, SHA-1 či SHA-2. Jeho slabinou je snadné nalezení pevných bodů, viz sekce 4.1.1.

Pro naše účely předpokládáme, že kompresní funkce F může být konstruována libovolně tak, aby byla bezkolizní a odolná proti nalezení vzoru a druhého vzoru.

1.2.1 Poznámka ke složitosti

Není-li řečeno jinak, je v tomto textu myšlen pod složitostí počet volání kompresní funkce při nezanedbatelné pravděpodobnosti úspěchu útoku. Ta je při používání narozeninového útoku přibližně 40% (viz sekce 2.1). Chceme-li se tedy přiblížit s pravděpodobností úspěchu útoku k jedné, potřebujeme zhruba 2,5-násobek uvedeného počtu volání. Tuto konstantu však pro přehlednost dále neuvádíme.

Někdy je uváděna pouze asymptotická složitost. Skutečná složitost $g(n)$ se chová jako $\mathcal{O}(f(n))$, pokud

$$\lim_{n\to\infty} \left|\frac{g(n)}{f(n)}\right| < +\infty\,.$$

1.3 Použití

Použití hašovacích funkcí je velmi rozmanité. V kryptografii se klade důraz zejména na bezkoliznost funkce a odolnost proti hledání druhého vzoru, v dalších aplikacích také na rychlost hašování. Vyjmenujme některé aplikace:

- Kontrola integrity dat – k porovnání shodnosti souborů či databází stačí srovnat pouze jejich haše.
- Autentizace dat – hašovací funkci využívá Keyed-hash Message Authentization Code (HMAC) ke kontrole, jestli nebyla zpráva během přenosu změněna.
- Digitální podpisy – díky odolnosti proti hledání druhého vzoru můžeme místo celého dokumentu podepisovat pouze jeho haš, což je mnohem snazší.

- Ověřování a ukládání hesel – v databázích se ukládají místo samotných hesel jejich otisky, což zabraňuje vyzrazení hesla. Stejně tak se během přihlašování přenáší po síti pouze haš hesla, která je poté porovnávána s uloženou hodnotou.
- Identifikace dat nebo souborů – otisk díky vlastnostem funkce téměř jednoznačně určuje daný soubor.
- Hašovací tabulky – hašovací funkce chovající se jako náhodné orákulum rovnoměrně distribuuje klíče hašovací tabulky.
- Generátory pseudonáhodných čísel – chování hašovací funkce nám umožňuje z malého počátečního vstupu (seed) získat velké množství „náhodných“ dat.
- Prokazování znalosti nebo autorství – zveřejněním haše dokumentu můžeme prokázat jeho znalost, aniž bychom jej uveřejnili.
- Derivace klíčů – haše mají mnohem větší entropii než například samotná uživatelská hesla, a proto se lépe hodí jako klíče pro šifrování.

Uveďme na příkladu, jak jsou pro aplikace důležité vlastnosti funkce. Na obrázku 1.2 jsou dva dokumenty se stejnou haší (to je umožněno zdánlivě bezvýznamnými znaky v textu reprezentovanými hvězdičkami). Jeden z nich by mohl být podstrčen k digitálnímu podpisu a podle druhého (falešného) dokumentu by později mohla být vyžadována větší suma k zaplacení, protože má stejnou haš, a tudíž by se jevil jako řádně podepsaný.

****
CONTRACT
At the price of **$176,495** Alf Blowfish
sells his house to Ann Bonidea. .....

****
CONTRACT
At the price of **$276,495** Alf Blowfish
sells his house to Ann Bonidea. .....

Obrázek 1.2: V roce 1996 H. Dobbertin prezentoval na konferenci FSE 1996 metodu nalézání kolizí u algoritmu MD4 [8].

Kapitola 2

Pomocná tvrzení

V této kapitole uvedeme několik tvrzení a výsledků z diskrétní pravděpodobnosti, teorie náhodných grafů a představíme dva kryptografické nástroje – Floydův algoritmus pro zacyklení a Hellmanovy tabulky.

2.1 Narozeninový paradox

Vyberme k prvků, jež náhodně nabývají n různých hodnot. Jaká je pravděpodobnost $p_1(k,n)$, že mezi těmito prvky bude alespoň jedna dvojice stejných:

$$\begin{aligned} p_1(k,n) &= 1 - \frac{n(n-1)\cdot\ldots\cdot(n-k+1)}{n^k} \\ &= 1 - \left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\cdot\ldots\cdot\left(1-\frac{k-1}{n}\right). \end{aligned}$$

V případě hašovacích funkcí je n velké, můžeme proto využít, že $\exp x \doteq 1+x$ pro x malé:

$$p_1(k,n) \doteq 1 - e^{-\frac{1}{n}} e^{-\frac{2}{n}} \cdot\ldots\cdot e^{-\frac{k-1}{n}} = 1 - e^{-\frac{1+2+\cdots+(k-1)}{n}} = 1 - e^{-\frac{k(k-1)}{2n}}.$$

Požadujeme-li tedy pravděpodobnost výskytu kolize p_1, volíme

$$k \doteq \sqrt{2\ln\left(\frac{1}{1-p_1}\right)}\sqrt{n}. \tag{2.1}$$

Pro $k \doteq \sqrt{n}$ je pravděpodobnost výskytu kolize přibližně 40%. Úloha je známa jako narozeninový paradox, neboť nám dává odpověď na otázku, kolik se musí sejít lidí, abychom měli 50% pravděpodobnost, že mezi budou dva narození ve stejný den v roce (předpokládáme rovnoměrné rozdělení dat narození během roku). Dosazením do výše odvozeného vzorce zjistíme, že dostačujících je 23 lidí. To je mnohem méně, než bychom si intuitivně mysleli. Většina lidí by očekávala něco kolem 183 lidí – automaticky totiž uvažují, kolik je potřeba lidí, aby mezi nimi s 50% pravděpodobností byl člověk, který má narozeniny ve stejný den jako oni, což odpovídá hledání druhého vzoru. Pravděpodobnost nalezení konkrétního prvku při k výběrech je

$$p_2(k,n) = 1 - \frac{(n-1)^k}{n^k} \doteq 1 - e^{-\frac{k}{n}}.$$

Úpravou získáme

$$k \doteq n \cdot \ln\left(\frac{1}{1-p_2}\right) . \tag{2.2}$$

Pro $p_2 = \frac{1}{2}$ je $k \doteq n \cdot \ln 2 \doteq 0,693n$. Aplikujeme-li výsledek na narozeniny, pak je potřeba 253 lidí, abychom s 50% pravděpodobností mezi nimi nalezli člověka s daným datem narození. Zvolíme-li $k \doteq n$ je $p_2 = 1 - e^{-1} \doteq 0,632$.

2.1.1 Mezi dvěma skupinami

Jaká je pravděpodobnost $p(k,l,n)$, že se mezi dvěma skupinami o k a l prvcích vyskytne alespoň jeden prvek, jenž bude v obou. Prvky mohou náhodně nabývat n různých hodnot. Pro případy hašovacích funkcí předpokládejme $k \ll n$ a $l \ll n$.

$$p(k,l,n) = 1 - \frac{n^k(n-k)^l}{n^{k+l}} = 1 - \left(1 - \frac{k}{n}\right)^l \doteq \frac{k}{n}\binom{l}{1} = \frac{k \cdot l}{n} . \tag{2.3}$$

Chceme-li tedy dosáhnout pravděpodobnosti výskytu kolize blížící se 1, musí platit:

$$k \ \cdot l \doteq n .$$

2.2 Náhodné grafy

V tomto odstavci uvedeme několik základních faktů z teorie náhodných grafů [9], které jsou potřeba při odvozování složitostí tvorby některých struktur.

Grafem G nazveme dvojici $G = (V, E)$, kde V je množina vrcholů grafu a E jeho hran. Vrcholy u a v spolu sousedí, pokud hrana $(u, v) \in E$. Cestou nazveme k-tici vrcholů $(v_1, \ldots, v_k)$ takovou, že pro všechna i, j různá platí $v_i \neq v_j$ a v_i, v_{i+1} jsou sousední pro $1 \leq i \leq k-1$. Graf se nazývá souvislý, vede-li z každého vrcholu cesta do všech ostatních. Párování je taková množina hran, ve které žádné dvě hrany nevedou do téhož vrcholu. Maximální párování, tedy to s nejvíce hranami, označíme M. Jestliže M pokrývá všechny vrcholy, jedná se o perfektní párování (perfect, one-factor matching). Graf $G' = (V', E')$ je izomorfní grafu $G = (V, E)$, pokud existuje bijekce $f : V \to V'$ taková, že $(u, v) \in E \Leftrightarrow (f(u), f(v)) \in E'$.

2.2.1 Prahová funkce

P. Erdös a A. Rényi [10] se zabývali pojmem náhodný graf $G = (V, p)$, kde V je opět množina vrcholů a p je pravděpodobnost, se kterou je každá dvojice vrcholů opatřena hranou. Počet vrcholů označíme ν. Charakteristiku grafu, která se nemění při izomorfních transformacích grafu, nazveme vlastnost (property). Jestliže se tato vlastnost zachovává při libovolném přidávání hran, hovoříme o monotónní vlastnosti.

Zda má graf monotónní vlastnost P určíme pomocí prahové funkce (threshold function). Funkce $t(\nu)$ je prahovou funkcí monotónní vlastnosti P, pokud pro $p \ll t(\nu)$ nemá graf vlastnost P téměř nikdy a pro $p \gg t(\nu)$ má graf vlastnost P téměř jistě.

Například pro perfektní párování je prahová funkce

$$t(\nu) = \frac{\ln(\nu)}{\nu} .$$

2.3 Floydův algoritmus pro zacyklení

V knize [16, kap. 3.2.2] je popsán algoritmus pro nalezení cyklu náhodné funkce $f : S \to S$, kde S je konečná množina mohutnosti N. Nechť x_0 je náhodná počáteční hodnota. Definujeme-li posloupnost $x_0, x_1, x_2, \ldots$ předpisem $x_{i+1} = f(x_i)$ pro $i \geq 1$, musí z konečnosti S dojít k jejímu zacyklení. Posloupnost se tedy skládá z ocasu (tail) očekávané délky $\sqrt{8\pi N}$ následovaného cyklem očekávané délky $\sqrt{8\pi N}$. Zacyklení bychom mohli najít prostým ukládáním hodnot x_i a jejich porovnáváním s již vypočtenými. Paměťové nároky jsou ale $\mathcal{O}(\sqrt{N})$. Floydův algoritmus umí tyto paměťové nároky podstatně snížit.

Podle Floyda začneme s dvojicí (x_1, x_2) a pak iterativně počítáme (x_i, x_{2i}) z (x_{i-1}, x_{2i-2}) pomocí f, případně f^2, dokud nenastane $x_m = x_{2m}$ pro nějaké m.

Nechť má ocas délku λ a cyklus μ, pak určitě $\lambda < m$ a $m = l \cdot \mu$ pro nějaké $l \in \mathbb{N}$. Odtud $l > \frac{\lambda}{\mu}$, a protože požadujeme co nejmenší m, tak $l = (1 + \lfloor \lambda/\mu \rfloor)$. Rovnost tedy poprvé nastane, když $m = \mu(1 + \lfloor \lambda/\mu \rfloor)$. Odtud platí následující nerovnost:

$$\lambda < m = \mu(1 + \lfloor \lambda/\mu \rfloor) \leq \mu + \lambda\,.$$

Z té plyne výpočetní složitost $\mathcal{O}(\sqrt{N})$, neboť μ i λ jsou $\mathcal{O}(\sqrt{N})$. Paměťové nároky jsou minimální, protože si stačí pamatovat kromě aktuální jen předchozí dvojici (x_{i-1}, x_{2i-2}).

2.4 Hellmanovy tabulky

V článku [11] M. E. Hellman popisuje způsob, jak najít vzor náhodné funkce $f : S \to S$, kde $\#S = N$, za $N^{\frac{2}{3}}$ volání funkce f a paměťové náročnosti $N^{\frac{2}{3}}$ za předpokladu předpočítání se složitostí N. Během předpočítání vytvoříme tabulky, které pak následně využijeme v online fázi hledání vzoru.

2.4.1 Použití Hellmanovy tabulky

Tabulka pro náhodnou funkci $f : S \to S$ se konstruuje následujícím způsobem:

1. Náhodně zvolíme m počátečních bodů (starting points) $\mathrm{SP}_1, \ldots, \mathrm{SP}_m$ z S a označíme $\mathrm{SP}_i =: x_{i0}$.
2. Pro $i \in \widehat{m}$ vypočteme: $x_{ij} = f(x_{i(j-1)})$, $j \in \widehat{t}$. Parametr t určuje délku řetězce.
3. Označíme koncové body (end points) $\mathrm{EP}_i := x_{it}$.
4. Zahodíme mezivýsledky a setřídíme $\{\mathrm{SP}_i, \mathrm{EP}_i\}_{i=1}^{m}$ podle EP_i.

Platí tedy, že $f^t(\mathrm{SP}_i) = \mathrm{EP}_i$ a matici obrazů f si můžeme představit takto:

$$\begin{array}{ccccccccc}
\mathrm{SP}_1 = x_{10} & \xrightarrow{f} & x_{11} & \xrightarrow{f} & x_{12} & \xrightarrow{f} & \ldots & \xrightarrow{f} & x_{1t} = \mathrm{EP}_1 \\
\vdots & & & & & & & & \\
\mathrm{SP}_i = x_{i0} & \xrightarrow{f} & x_{i1} & \xrightarrow{f} & x_{i2} & \xrightarrow{f} & \ldots & \xrightarrow{f} & x_{it} = \mathrm{EP}_i \\
\vdots & & & & & & & & \\
\mathrm{SP}_m = x_{m0} & \xrightarrow{f} & x_{m1} & \xrightarrow{f} & x_{m2} & \xrightarrow{f} & \ldots & \xrightarrow{f} & x_{mt} = \mathrm{EP}_m
\end{array}$$

Mějme nyní zprávu $Y = Y_1$, k níž chceme najít vzor X, $f(X) = Y_1$. Iterujeme $Y_j = f(Y_{j-1})$, dokud nenastane $Y_j = \mathrm{EP}_{i_0}$ pro nějaké $i_0 \in \widehat{m}$, nebo $j = t+1$. V prvním případě spočteme $\widetilde{X} = f^{t-j}(\mathrm{SP}_{i_0})$ a ověříme, že $f(\widetilde{X}) = Y$. Pokud ano, je $\widetilde{X}$ hledaným vzorem X. Jinak se jedná o tzv. falešný poplach. Může se totiž stát, že nějaké $x_{i_0 k}$ má více vzorů a iterováním jsme se připojili do řetězce „z boku“. Jestliže nastane falešný poplach nebo dojdeme až k $j = t+1$, tabulka nepokrývá hodnotu vzoru Y.

2.4.2 Pokrytí hodnot tabulkou

Pokud by všechny hodnoty x_{ij} v tabulce byly různé, pravděpodobnost, že zadanou hodnotu v tabulce nalezneme, by byla $\frac{mt}{N}$. Funkce f je ale náhodná, proto může docházet k překryvu hodnot a pravděpodobnost p_{in}, že se daná hodnota nachází v tabulce, je podle věty v Hellmanově článku [11] omezena zdola takto:

$$p_{in} \geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(\frac{N - it}{N} \right)^{j+1} . \tag{2.4}$$

Vezmeme-li N velké a pevné, můžeme ze vztahu (2.4) odvodit následující poznatky.

- Zvyšování m a t více než $mt^2 = N$ nemá velký význam, neboť nahradíme-li $(1 - \frac{it}{N})^j \doteq \exp(-\frac{ijt}{N})$, pak poslední člen sumy je přibližně $\exp(-\frac{mt^2}{N})$. Příspěvek k omezení pravděpodobnosti je tedy pro $mt^2 \gg N$ velmi malý.
- Naopak pro $mt^2 \ll N$ jsou sčítance v sumě přibližně rovny 1, a tudíž je pravá strana rovna přibližně $\frac{mt}{N}$. To je ale zároveň horní odhad p_{in}. Společně s předchozí poznámkou tedy vidíme, že má smysl volit m a t tak, aby platilo $mt^2 = N$.
- Nyní zkusíme odhadnout dolní mez p_{in} ze vztahu (2.4) pro $m = t$ a $mt^2 = N$ pomocí výrazu $e^x \doteq 1 + x$ a horních mezí:

$$\begin{aligned} p_{in} &\geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(\frac{N - it}{N} \right)^{j+1} \doteq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \exp\left(-\frac{ijt}{N} \right) \geq \\ &\geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \exp\left(-\frac{jtm}{N} \right) = \frac{mt}{N} \cdot \frac{1 - \exp\left(-\frac{mt^2}{N}\right)}{\left(1 - \exp\left(-\frac{mt}{N}\right)\right) t} \doteq \\ &\doteq \frac{mt}{N} \cdot \frac{1 - \exp(-1)}{\frac{1}{t} \cdot t} = \frac{mt}{N} \cdot \left(1 - e^{-1}\right) \doteq 0,63 \cdot \frac{mt}{N} . \end{aligned}$$

  Numericky lze podle [11] dospět dokonce k $p_{in} \geq 0{,}80 \cdot \frac{mt}{N}$.

Pomineme-li faktor 0,80 před zlomkem, je pro $m = t$ a $mt^2 = N$ pravděpodobnost $p_{in} \geq N^{-\frac{1}{3}}$. Potřebujeme tedy zhruba $d = N^{\frac{1}{3}}$ nezávislých tabulek, abychom s dostatečnou pravděpodobností mohli hledat vzor. Pro dosažení nezávislosti použijeme v i-té tabulce místo f funkci $f_i(x) := f(x \oplus i)$. Tímto máme zaručeno, že se v případě stejné hodnoty v různých tabulkách nebude opakovat celý řádek, a zároveň snadno ze vzoru f_i nalezneme vzor f.

Dále musíme ošetřit falešné poplachy. Podle věty v [11] je průměrný počet falešných poplachů na jednu tabulku $E(F)$ omezen následovně

$$E(F) \leq \frac{mt(t-1)}{2N} \leq \frac{mt^2}{2N} ,$$

což pro $mt^2 = N$ dává $E(F) \leq \frac{1}{2}$. Falešný poplach tedy nastane v průměru u každé druhé tabulky, navýší proto potřebný počet tabulek maximálně dvakrát.

Pro nalezení vzoru X ke zprávě Y postupně zkoušíme tabulky. V případě falešného poplachu nebo nenalezení hodnoty pokračujme další tabulkou. Pro $m = t = N^{\frac{1}{3}}$ a počet tabulek $d = 2 \cdot N^{\frac{1}{3}}$ bychom měli s velkou pravděpodobností vzor v jedné z tabulek najít.

2.4.3 Složitost tvorby

Během předpočítání musíme vytvořit d tabulek o m řetězcích, přičemž na každý z nich potřebujeme t operací. Odtud je složitost předpočítání $m \cdot t \cdot d = N$. Potřebná paměť pro uložení tabulek je $\mathcal{O}(m \cdot d) = \mathcal{O}(N^{\frac{2}{3}})$. Během hledání vzoru procházíme d tabulek a iterujeme nejvýše t-krát, z čehož plyne $\mathcal{O}(t \cdot d) = \mathcal{O}(N^{\frac{2}{3}})$ volání funkce f.

Kapitola 3

Hledání kolizí a multikolizí

Ditherování – metoda, o kterou se v této práci zajímáme nejvíce – sice nezvýší odolnost hašovací funkce proti hledání kolizí a multikolizí, nicméně tyto dva útoky jsou základem pro hledání druhého vzoru, proto je zde popíšeme. Jouxův útok pro hledání multikolizí nám také dává překvapivý výsledek o řetězení hašovacích funkcí.

3.1 Hledání kolizí

V této sekci se budeme věnovat hledání kolizí, což je dvojice zpráv M a M' taková, že $M \neq M'$ a zároveň $f(M) = f(M')$. Hledání kolizí je nejsnazším útokem na hašovací funkce a zároveň je základem pro většinu dalších útoků.

3.1.1 Narozeninový útok

Nejprimitivnějším způsobem je útok hrubou silou neboli narozeninový útok. Podle sekce 2.1 bychom měli po vyzkoušení přibližně $2^{\frac{n}{2}}$ zpráv nalézt kolizi dvou zpráv. Nemůžeme ovšem ovlivnit tvar těchto dvou zpráv.

3.1.2 Yuvalův útok

Yuval nevylepšuje výpočetní složitost hledání kolizí, pouze dává návod, jak do jisté míry ovlivnit podobu zpráv [16, str. 369-370]. Označme legální zprávu x_l a její podvrh x_p. Počet bloků zpráv x_l a x_p označme ℓ. Podstatou útoku jsou drobné modifikace zpráv značené x'_l, x'_p spočívající v malých změnách x_l a x_p, například výměně netisknutelných znaků. Dále:

1. Vypočteme haše $2^{\frac{n}{2}}$ modifikací legální zprávy x'_l a ukládáme je do hašovací tabulky.
2. Počítáme haše drobných modifikací podvržené zprávy x'_p, dokud nedojde ke shodě. K té dojde podle sekce 2.1 přibližně po $2^{\frac{n}{2}}$ voláních.

Provedeme $2^{\frac{n}{2}+1}$ volání hašovací funkce, tedy složitost celého útoku je $\ell \cdot 2^{\frac{n}{2}+1}$ volání kompresní funkce, paměťové nároky $\mathcal{O}(2^{\frac{n}{2}})$.

3.1.3 Varianta s malou paměťovou náročností

Pomocí Floydova algoritmu popsaného v sekci 2.3 umíme najít kolizi náhodné mapovací funkce $r : \mathsf{H} \to \mathsf{H}$. Nechť je tedy $g_x(h) = x'$ funkce přiřazující haši h zprávu x' délky ℓ bloků, která je drobnou modifikací zprávy x (např. jednotlivé bity H určují, zda se na dané pozici má zpráva x změnit). Definujeme:

$$r(h) = \begin{cases} f(g_{x_l}(h)) & \text{pro } \mathrm{lsb}(h) = 0 \\ f(g_{x_p}(h)) & \text{pro } \mathrm{lsb}(h) = 1\,, \end{cases}$$

kde $\mathrm{lsb}(h)$ značí nejméně významný bit h.

Máme tedy náhodnou funkci na prostoru haší, pro kterou můžeme pomocí Floydova algoritmu nalézt kolizní dvojici h_1, h_2, která se s 50% pravděpodobností liší paritou. Pak můžeme získat kolizi zpráv x'_l a x'_p jako $x'_l = g_{x_l}(h_1)$ a $x'_p = g_{x_p}(h_2)$.

Tento útok má výpočetní složitost $\mathcal{O}(\ell \cdot 2^{\frac{n}{2}})$, ale paměťové nároky jsou zanedbatelné.

3.2 Jouxův útok

Multikolizí nebo s-kolizí máme na mysli s zpráv $M^{(1)}, M^{(2)}, \ldots, M^{(s)}$ takových, že $f(M^{(1)}) = f(M^{(2)}) = \cdots = f(M^{(s)})$. K. Suzuki a spol. ukázali v článku [22], že pro náhodnou funkci f potřebujeme k nalezení s-kolize s pravděpodobností přibližně 1/2 vyzkoušet $(s!)^{\frac{1}{s}} \cdot 2^{\frac{n(s-1)}{s}}$ zpráv.

Iterované hašovací funkce výše zmíněné teoretické složitosti zdaleka nedosahují, neboť A. Joux představil útok [12], jak nalézt 2^k-kolizi se složitostí $k \cdot 2^{\frac{n}{2}}$ místo teoretické hodnoty $(2^k!)^{\frac{1}{2^k}} \cdot 2^{\frac{n(2^k-1)}{2^k}}$ volání kompresní funkce.

Podstata útoku je následující – pomocí narozeninového paradoxu či jiného útoku na kolizi nacházíme páry bloků M_i a M'_i takových, že

$$F(h_{i-1}, M_i) = F(h_{i-1}, M'_i)\,.$$

Nalezneme-li těchto dvojic k pro různá i, můžeme na k místech ve zprávě M volit ze dvou různých bloků, a tedy vytvořit 2^k rozdílných zpráv se stejnou haší. Všimněme si, že tyto zprávy mají dokonce shodné všechny průběžné haše.

Jouxův útok ukazuje, že zřetězením $f(M)||g(M)$ dvou různých hašovacích funkcí f a g produkujících haše délek n_f a n_g se nezlepší odolnost vůči hledání kolizí podle očekávání na $\mathcal{O}(2^{\frac{n_f+n_g}{2}})$. Kolizi lze totiž najít za $n_g 2^{\frac{n_f}{2}-1}$ volání kompresní funkce F a $2^{\frac{n_g}{2}}$ volání hašovací funkce g.

Pomocí Jouxova útoku nalezneme $2^{\frac{n_g}{2}}$-kolizi hašovací funkce f se složitostí $n_g 2^{\frac{n_f}{2}-1}$. Pak spočteme hodnoty funkce g pro zprávy z multikolize. Těch je dostatečné množství, aby mezi nimi existovala kolize, která je zároveň kolizí hašovací funkce vzniklé zřetězením. Poznamenejme ještě, že konstrukce funkce g může být libovolná, ne nutně iterativní. Řetězení dvou hašovacích funkcí, z nichž jedna je iterativní, je tedy přibližně stejně odolné proti kolizi jenom jako silnější z funkcí.

Kapitola 4

Hledání druhého vzoru

Iterativní konstrukce hašovací funkce nám umožňuje hledat druhý vzor ke zprávě M_{target} nalezením začátku zprávy takového, že se jeho haš bez aplikace Merkleova-Damgårdova zesílení rovná některé z průběžných haší zprávy M_{target}. Poté připojíme k začátku odpovídají konec zprávy M_{target} a upravíme celkovou délku zprávy tak, abychom splnili Merkleovo-Damgårdovo zesílení. K úpravě délky použijeme rozšiřitelnou zprávu nebo kolizní strom.

4.1 Útok pomocí rozšiřitelné zprávy

J. Kelsey a B. Schneier [14] vymysleli útok pro naleznení druhého vzoru využívající rozšiřitelné zprávy, což je struktura umožňující produkovat zprávy délky v daném rozmezí se stejnou poslední průběžnou haší. Tato zpráva nezahrnuje Damgårdovo-Merkleovo zesílení, proto neprodukuje přímo kolize zpráv, ale můžeme ji použít k hledání druhého vzoru napojením na určité místo cílové zprávy.

Při tvorbě rozšiřitelné zprávy můžeme využít faktu, že pro funkce typu MD5, SHA či Snefru umíme nalézt pevné body, tedy blok zprávy M a vstupní haš h takové, že $h = F(h, M)$.

S jen o něco málo vyšší složitostí umíme vytvořit rozšiřitelnou zprávu i bez pevných bodů využitím myšlenky Jouxova útoku na multikolizi (viz sekce 3.2).

4.1.1 Hledání pevných bodů

Hašovací funkce typu MD5 nebo SHA jsou založeny na Daviesově-Meyerově principu s využitím blokové šifry $y = E(k, x)$, kde k je klíč a x šifrovaná zpráva. Kompresní funkce F potom vypadá následovně:

$$F(h, M) = E(M, h) \oplus h\,,$$

tedy jako klíč k použijeme blok zprávy M a pomocí něj šifrujeme průběžnou haš h. Blokovou šifru umíme se zadaným klíčem invertovat, $x = E^{-1}(k, y)$. Pro námi zvolený blok M spočteme právě jeden pevný bod h takto:

$$\begin{aligned} h = F(h, M) &= E(M, h) \oplus h \\ 0 &= E(M, h) \\ h &= E^{-1}(M, 0)\,. \end{aligned} \tag{4.1}$$

4.1.2 Tvorba rozšiřitelné zprávy pomocí pevných bodů

Tvorba rozšiřitelné zprávy $M(\ell)$, kde ℓ je požadovaný počet bloků zprávy, probíhá následovně:

1. Najdeme $2^{\frac{n}{2}}$ pevných bodů (h_i, M_{fix}). Bloky M_{fix} volíme a haš k nim dopočteme podle vztahu (4.1). Každý blok má právě jeden pevný bod.
2. Vypočteme $2^{\frac{n}{2}}$ haší $h'_j = f(IV, M_1)$ tak, že bereme náhodně bloky M_1.
3. Mezi těmito dvěma seznamy najdeme kolizi podle sekce 2.1. Společnou haš označíme h_{exp}.
4. Vytvoříme rozšiřitelnou zprávu $M(\ell)$:
$$M(\ell) := M_1 || M_{fix}^{\ell-1}\,,$$
$$f(IV, M(\ell)) = h_{exp}\,.$$

Tento proces vyžaduje $2^{\frac{n}{2}+1}$ volání kompresní funkce.

4.1.3 Obecná tvorba rozšiřitelné zprávy

Pro tvorbu rozšiřitelné zprávy potřebujeme umět hledat kolize zpráv různé délky, konkrétně α a 1, se stejnou vstupní haší h_{in}. To provedeme následovně:

1. Vypočteme haš $h_\alpha = f(h_{in}, M(\alpha - 1))$, kde $M(\alpha - 1)$ je libovolně pevně zvolená zpráva délky $\alpha - 1$.
2. Vytvoříme seznam haší $h_i = F(h_\alpha, M_i)$ pro $i \in \widehat{2^{\frac{n}{2}}}$, kde M_i jsou libovolně volené bloky.
3. Vytvoříme seznam haší $h'_i = F(h_{in}, M'_i)$ pro $i \in \widehat{2^{\frac{n}{2}}}$, kde M'_i jsou libovolně volené bloky.
4. Mezi těmito seznamy nalezneme podle vztahu (2.3) kolizi $h_j = h'_k$.
5. Kolizní zprávy jsou M'_k a $M(\alpha - 1)||M_j$.

K nalezení této kolize potřebujeme $\alpha - 1 + 2^{\frac{n}{2}+1}$ volání kompresní funkce.

Nyní můžeme sestrojit rozšiřitelnou zprávu:

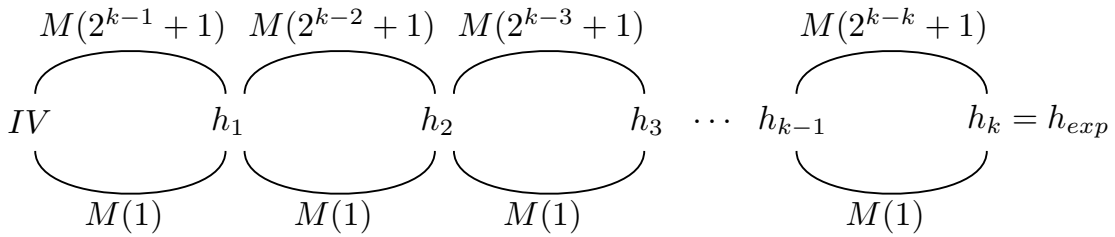
1. Zvolíme si počáteční vstupní haš IV.
2. Podle předchozího algoritmu najdeme kolizní zprávy délky 1 a $2^{k-i} + 1$ pro $i \in \widehat{k}$ tak, aby pro pár $i = 1$ byla použita jako vstupní haš IV a dále jako vstupní haš i-tého páru výstupní haš $(i-1)$-ho páru.
3. Výstupní haš k-tého páru označíme h_{exp}.

Nalezené zprávy můžeme v daném pořadí řetězit za sebe a získat tak zprávy v rozmezí délek k až $k + 2^k - 1$ se stejnou výslednou haší h_{exp} (viz obrázek 4.1). Chceme-li rošiřitelnou zprávu $M(\ell)$ délky ℓ, provedeme binární rozvoj $\ell - k$ a doplníme jej zleva nulami, tak aby měl délku k. Nyní, je-li na i-té pozici 1, volíme v i-tém páru kolizní zprávu délky $2^{k-i} + 1$, v případě 0 zprávu délky 1.

Složitost tvorby je

$$\sum_{i=1}^{k}(2^{k-i} - 1 + 2^{\frac{n}{2}+1}) = k \cdot 2^{\frac{n}{2}+1} - k + \sum_{i=0}^{k-1} 2^i = k \cdot 2^{\frac{n}{2}+1} + 2^k - (k+1)\,.$$

Pro $k < \frac{n}{2}$ je podstatná část $k \cdot 2^{\frac{n}{2}+1}$.



Obrázek 4.1: Schéma obecné rozšiřitelné zprávy.

4.1.4 Hledání druhého vzoru

Pomocí vytvořené rozšiřitelné zprávy $M(\ell)$ nyní nalezneme 2. vzor zprávy M_{target} délky $2^k + k + 1$, případně $2^k + 3$ bloků, pokud použijeme rozšiřitelnou zprávu z pevných bodů. Jednotlivé bloky označme M_i.

1. Spočteme průběžné haše h_i pro M_{target} a uložíme je do nějaké snadno prohledávatelné struktury, např. hašovací tabulky.

2. Hledáme zprávy M_{link} takové, že $F(h_{exp}, M_{link}) = h_j$ pro nějaké $k < j < 2^k + k + 1$ (v případě pevných bodů $2 < j < 2^k + 3$). Podle sekce 2.1 musíme zkusit přibližně 2^{n-k} bloků.

3. Nastavíme rozšiřitelnou zprávu tak, aby byla j bloků dlouhá.

4. Vytvoříme 2. vzor:

$$M_{\text{second}} := M(j)||M_{link}||M_{j+1}||\dots||M_{2^k+k+1}\,.$$

   Platí

$$f(IV, M_{\text{second}}) = f(IV, M_{\text{target}})\,.$$

Náročnost hledání druhého vzoru včetně tvorby rozšiřitelné zprávy je $k \cdot 2^{\frac{n}{2}+1} + 2^{n-k}$ volání kompresní funkce pro obecnou rozšiřitelnou zprávu a pouze $2^{\frac{n}{2}+1} + 2^{n-k}$ pro rozšiřitelnou zprávu vytvořenou pomocí pevných bodů.

4.1.5 Druhý vzor s libovolným prefixem

Obecnou rozšiřitelnou zprávu i z pevných bodů můžeme vytvořit také s libovolným prefixem X tak, že během tvorby rozšiřitelné zprávy místo IV při výpočtu haší použijeme $f(IV, X)$. Stejně tak můžeme za rozšiřitelnou zprávu připojit zprávu Y a pro hledání M_{link} v algoritmu hledání druhého vzoru použít jako vstupní haš $f(h_{exp}, Y)$ místo h_{exp}. Minimální délka rozšiřitelné zprávy vytvořené pomocí pevných bodů je pak $l_X + l_Y + 2$, případně $l_X + l_Y + k$ pro obecnou rozšiřitelnou zprávu, kde l_X, l_Y značí počet bloků zpráv X a Y. Rozšiřitelná zpráva pak vypadá následovně:

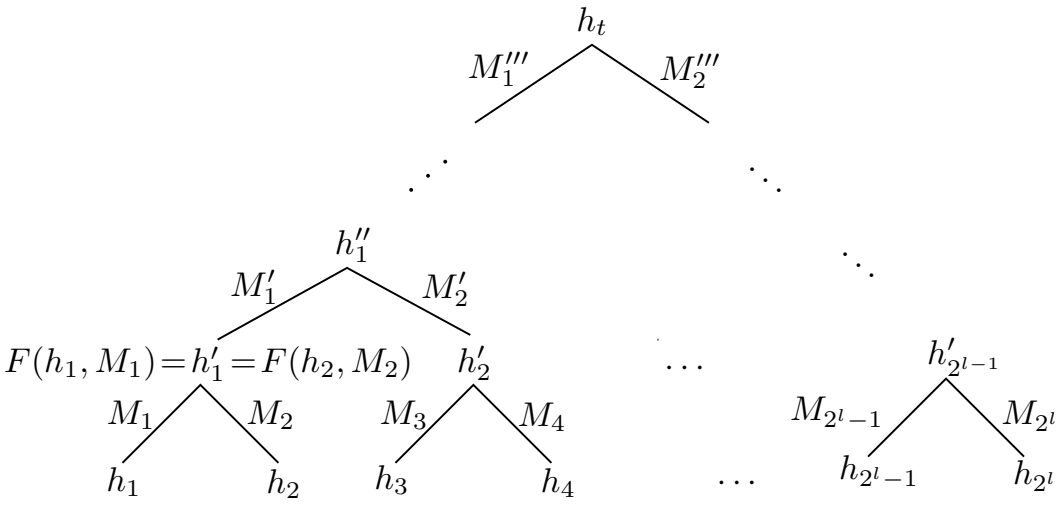
$$M_{XY}(\ell) := X||M_1||M(\ell - 1 - l_X - l_Y)||Y\,.$$

4.2 Útok pomocí kolizního stromu

V článku [1] popisuje A. Shamir a spol. útok pro hledání druhého vzoru pomocí kolizního stromu. Ten byl popsán J. Kelseym a T. Kohnem v článku [13]. Útok funguje jak na klasickou Merkleovu-Damgårdovu konstrukci, tak na ditherované hašovací funkce. Hlavní myšlenkou je napojit strukturu zvanou kolizní strom (collision tree, diamond structure), na cílovou zprávu a pak ke stromu připojit prefix patřičné délky. Složitost útoku byla v článku A. Shamira při použitém algoritmu uvedena chybně jako $\mathcal{O}\left(2^{\frac{n+l}{2}} + 2^{n-k} + 2^{n-l}\right)$, na hodnotu $\mathcal{O}\left(\sqrt{l} \cdot 2^{\frac{n+l}{2}} + 2^{n-k} + 2^{n-l}\right)$ ji opravil S. R. Blackburn, D.R. Stinson a J. Upadhyay v článku [5]. Finská skupina autorů kolem J. Kortelainena ovšem nedávno nalezla jiný algoritmus konstrukce kolizního stromu, který má složitost stejnou, jako byla původně uvedená. V této sekci popíšeme původní algoritmus s opravenou složitostí.

4.2.1 Kolizní strom

Kolizní strom je dokonale vyvážený binární strom, jehož uzly jsou značeny hašemi h_i délky n bitů a hrany bloky zpráv M_i délky m bitů. Z uzlu h_i do h_j vede orientovaná hrana M_r, pokud $h_i = F(h_j, M_r)$ (viz obrázek 4.2). Je-li počet hran z listu do kořene l, řekneme, že strom má hloubku l. Strom má 2^l listů, tedy $2^{l+1} - 1$ vrcholů. Haš v kořeni stromu označíme h_t. Máme tedy 2^l haší, které po l hašováních příslušnými zprávami vedou na stejnou haš h_t.



Obrázek 4.2: Schéma kolizního stromu.

Konstrukce kolizního stromu

Kolizní strom se konstruuje odspodu. Naivním přístupem bychom postupně pro dvojice haší hledali kolizi ($2^{\frac{n}{2}+i}$ volání pro i-tou úroveň). Výhodnější je ale hledat kolize mezi všemi zvolenými vstupními hašemi. Přechod z úrovně o 2^i listech na 2^{i-1} probíhá následujícím způsobem:

1. Vezmeme daných 2^i haší. (V nejnižší úrovni si je můžeme zvolit. Jako jednu z nich použijeme IV.)

2. Pro každou z haší spočteme pomocí $\lfloor\sqrt{\ln 2}\sqrt{i}\cdot 2^{\frac{n}{2}-\frac{i}{2}}\rfloor$ různých zpráv výsledné haše.

3. Pravděpodobnost kolize mezi dvěma seznamy haší je podle vztahu (2.3) rovna přibližně $L^2\cdot 2^{-n}$, kde L je počet haší v seznamu. Představíme-li si listy jako vrcholy náhodného grafu, je pravděpodobnost existence hrany mezi dvěma vrcholy (tzn. kolize haší)

$$p \doteq (\sqrt{\ln 2}\sqrt{i}\cdot 2^{\frac{n}{2}-\frac{i}{2}})^2\cdot 2^{-n} = \frac{\ln 2\cdot i\cdot 2^{n-i}}{2^n} = \frac{\ln 2^i}{2^i}\,.$$

   Počet vrcholů v grafu neboli počet listů označíme $\nu = 2^i$. Odtud $p = \frac{\ln\nu}{\nu}$, což je hodnota prahové funkce pro existenci perfektního párování v grafu. Jeho haše tvoří další patro kolizního stromu (viz sekce 2.2).

Tento postup zopakujeme l-krát pro tvorbu celého stromu. Složitost jednoho přechodu je $2^i\cdot\sqrt{\ln 2}\sqrt{i}\cdot 2^{\frac{n}{2}-\frac{i}{2}} = \sqrt{\ln 2}\sqrt{i}\cdot 2^{\frac{n}{2}+\frac{i}{2}}$. Horní odhad počtu volání kompresní funkce na celou konstrukci je:

$$\begin{aligned}
&\sqrt{\ln 2}\cdot 2^{\frac{n}{2}}\sum_{i=1}^{l}\sqrt{i}2^{\frac{i}{2}} < \sqrt{\ln 2}\cdot 2^{\frac{n}{2}}\sqrt{l}\sum_{i=1}^{l}2^{\frac{i}{2}} = \\
&= \sqrt{\ln 2}\cdot 2^{\frac{n}{2}}\sqrt{l}\cdot\frac{2^{\frac{l+1}{2}}-1}{2^{\frac{1}{2}}-1}\sqrt{2} \le \sqrt{2\ln 2}\cdot 2^{\frac{n}{2}}\sqrt{l}\cdot 2^{\frac{l+1}{2}}(\sqrt{2}+1) \le \\
&\le 4\sqrt{2\ln 2}\cdot\sqrt{l}\cdot 2^{\frac{n+l}{2}} \doteq 4{,}71\sqrt{l}\cdot 2^{\frac{n+l}{2}}\,.
\end{aligned}$$

Jako dolní odhad složitosti můžeme použít složitost přechodu z listového patra, tzn. $\sqrt{\ln 2}\sqrt{l}\cdot 2^{\frac{n}{2}+\frac{l}{2}} = 0{,}83\sqrt{l}\cdot 2^{\frac{n}{2}+\frac{l}{2}}$. Na konstrukci kolizního stromu tedy potřebujeme $\mathcal{O}(\sqrt{l}\cdot 2^{\frac{n}{2}+\frac{l}{2}})$ volání kompresní funkce.

Výpočetní složitost

U složitějších struktur, jako je například kolizní strom, může v případě reálné implementace hrát roli kromě volání hašovací funkce také vyhledávání párování, jak ukazuje S. R. Blackburn, D.R. Stinson a J. Upadhyay v článku [5]. Proto nyní podle jejich článku odvodíme celkovou složitost konstrukce kolizního stromu za předpokladu, že jedno volání kompresní funkce považujeme za jednu operaci.

Pro přechod z jednoho patra musíme:

1. Spočítat $L = \sqrt{\ln 2}\sqrt{i}\cdot 2^{\frac{n}{2}-\frac{i}{2}}$ haší pro 2^i listů, což dává složitost $N = 2^iL$.

2. Vytvořit z listů graf, tzn. najít kolize mezi hašemi. Asymptoticky toto nelze provést lépe než všechny haše seřadit a dalším průchodem nalézt shodné. To si vyžádá $\mathcal{O}(N\ln N)$ operací, tedy $\mathcal{O}(2^iL\ln(2^iL))$.

3. Najít v grafu perfektní párování. Podle [10] vyžaduje algoritmus pro hledání maximálního párování $\mathcal{O}(\frac{\varepsilon\ln\nu}{\ln\ln\nu})$ operací, přičemž $\varepsilon = \#E$ a $\nu = \#V$. Maximální párování bude v našem případě perfektní, neboť graf je podle sekce 2.2.1 téměř jistě obsahuje, protože pravděpodobnost existence hrany přesahuje prahovou funkci.

   Počet vrcholů je $\nu = 2^i$ a očekávaný počet hran

$$\varepsilon = 2^{-n}L^2\binom{2^i}{2} \doteq 2^{-n}\cdot i2^{n-i}\cdot\frac{2^i(2^i-1)}{2} \doteq i\cdot 2^i\,.$$

Dosazením získáme složitost

$$\mathcal{O}\left(\frac{i^2 2^i}{\ln i}\right).$$

Kombinací těchto tří kroků získáváme výpočetní složitost pro přechod z jednoho patra

$$\mathcal{O}\left(2^i L + 2^i L \ln(2^i L) + \frac{i^2 2^i}{\ln i}\right) = \mathcal{O}\left(2^i L \ln(2^i L) + \frac{i^2 2^i}{\ln i}\right).$$

Dosadíme-li $L = \mathcal{O}(\sqrt{i} \cdot 2^{\frac{n}{2}-\frac{i}{2}})$, dostaneme

$$\begin{aligned} 2^i L \ln(2^i L) &= \mathcal{O}\left(\sqrt{i} \cdot 2^{\frac{n}{2}+\frac{i}{2}} \cdot \ln\left(\sqrt{i} \cdot 2^{\frac{n}{2}+\frac{i}{2}}\right)\right) = \\ &= \mathcal{O}\left(\sqrt{i} \cdot 2^{\frac{n}{2}+\frac{i}{2}} \cdot \left(\frac{n+i}{2} + \frac{1}{2}\ln i\right)\right) = \\ &= \mathcal{O}\left(n \cdot \sqrt{i} \cdot 2^{\frac{n}{2}+\frac{i}{2}}\right) \end{aligned}$$

pro $n > i$.

Složitost konstrukce celého stromu získáme sečtením přes všechny úrovně kromě první, neboť mezi dvěma hašemi hledáme pouze kolizi, nikoliv párování:

$$\mathcal{O}\left(\sum_{i=2}^{l}\left[n \cdot \sqrt{i} \cdot 2^{\frac{n+i}{2}} + \frac{i^2 2^i}{\ln i}\right]\right).$$

První část sumy má stejné tvar jako v případě, kdy jsme počítali pouze volání kompresní funkce, vynásobený n. Druhou odhadneme takto:

$$\sum_{i=2}^{l} \frac{i^2 2^i}{\ln i} < \sum_{i=2}^{l} i^2 2^i < l^2 \sum_{i=2}^{l} 2^i = l^2 \cdot 2(2^l - 2) = \mathcal{O}\left(l^2 2^l\right).$$

Dohromady je celková složitost

$$\mathcal{O}\left(n \cdot \sqrt{l} \cdot 2^{\frac{n+l}{2}} + l^2 \cdot 2^l\right).$$

Pro $l = \alpha n, \alpha < 1$ platí

$$\lim_{n\to\infty} \frac{l^2 2^l}{n\sqrt{l} \cdot 2^{\frac{n+l}{2}}} \leq \lim_{n\to\infty} \sqrt{n} \cdot 2^{\frac{l-n}{2}} = \lim_{n\to\infty} \sqrt{n} \cdot 2^{\frac{(\alpha-1)n}{2}} = 0,$$

protože $\alpha - 1 < 0$. Druhý člen je tedy asymptoticky menší než první, proto můžeme brát složitost pouze

$$\mathcal{O}\left(n \cdot \sqrt{l} \cdot 2^{\frac{n+l}{2}}\right),$$

což je n-krát počet volání kompresní funkce.

Můžeme se také přesvědčit, že je druhý člen menší už pro $n \geq 6$ a $\alpha = \frac{3}{4}$. (V útocích je l většinou kolem $\frac{n}{2}$). Pak

$$\begin{aligned} n \cdot \sqrt{l} \cdot 2^{\frac{n+l}{2}} &\geq l^2 \cdot 2^l \\ n\sqrt{n}\sqrt{\alpha} \cdot 2^{(\frac{\alpha+1}{2})n} &\geq \alpha^2 n^2 \cdot 2^{\alpha n} \\ 2^{(\frac{1-\alpha}{2})n} &\geq \alpha^{\frac{3}{2}}\sqrt{n} \\ 2^{\frac{n}{8}} &\geq \frac{3\sqrt{3}}{8}\sqrt{n}. \end{aligned}$$

Pro $n = 6$

$$2^{\frac{6}{8}} \geq \frac{3\sqrt{3}}{8} \cdot \sqrt{6}$$
$$2^{\frac{1}{4}} \geq \frac{9}{8},$$

což můžeme numerickým vyčíslením ověřit, že platí. Předpokládejme nyní, že rovnice platí pro n, a ukážeme, že pro $n \geq 6$ platí i pro $n + 1$:

$$2^{\frac{n+1}{8}} \geq \frac{3\sqrt{3}}{8}\sqrt{n} \cdot 2^{\frac{1}{8}} \geq \frac{3\sqrt{3}}{8}\sqrt{n+1},$$

neboť

$$n \cdot 2^{\frac{1}{4}} \geq n + 1$$
$$n \geq \frac{1}{2^{\frac{1}{4}} - 1} \doteq 5,285.$$

4.2.2 Hledání druhého vzoru

Hledání druhého vzoru ke zprávě M_{target} délky $2^k + l + 1$ s průběžnými hašemi $h_1, \ldots, h_{2^k+l+1}$ probíhá podle tohoto algoritmu:

1. Zkonstruujeme kolizní strom hloubky l podle sekce 4.2.1, na což potřebujeme $\lfloor 4\sqrt{2 \ln 2} \cdot \sqrt{l} \rfloor \cdot 2^{\frac{n+l}{2}}$ volání kompresní funkce. Haše v listech si označíme $h'_1, \ldots, h'_{2^l}$, haš v kořeni h_t.

2. Hledáme blok M_{link} takový, že $F(h_t, M_{\text{link}}) = h_{i_0}$ pro nějaké i_0 splňující nerovnost $l + 1 \leq i_0 < 2^k + l + 1$. Podle sekce 2.1 musíme zkusit přibližně 2^{n-k} bloků pro nalezení M_{link}.

3. Pokud $i_0 = l + 1$, vezmeme za h'_{j_0} hodnotu IV. Jinak hledáme prefix P délky $i_0 - l - 1$, pro který platí $f(IV, P) = h'_{j_0}$, $j_0 \in \widehat{2^l}$. Potřebný počet volání kompresní funkce je 2^{n-l}.

4. Vytvoříme zprávu T tak, že při průchodu stromem z h'_{j_0} do h_t postupně řetězíme bloky z prošlých hran. Takto vzniklá zpráva má délku l.

5. Sestavíme druhý vzor M_{second}:

$$M_{\text{second}} := P||T||M_{\text{link}}||M_{i_0+1}|| \ldots ||M_{2^k+l+1}.$$

Celková složitost algoritmu je $\lfloor 4\sqrt{2 \ln 2} \cdot \sqrt{l} \rfloor \cdot 2^{\frac{n+l}{2}} + 2^{n-k} + 2^{n-l}$ volání kompresní funkce.

Kapitola 5

Kombinatorika na slovech

V této kapitole uvedeme základní pojmy z kombinatoriky na slovech [3]. Dále uvedeme věty, které posléze použijeme pro konstrukci vhodné ditherační posloupnosti. Co je to ditherování bude řečeno v další kapitole, prozatím je důležité, že vhodná ditherační posloupnost musí být square-free a mít exponenciální komplexitu. V závěru kapitoly zmíníme algoritmy pro generování prefixů nekonečných slov.

5.1 Základní pojmy

Nechť $\mathcal{A} = \{a_1, a_2, \ldots, a_k\}$ je konečná množina symbolů. Ty pak nazveme písmeny, množinu abecedou. Dále slovem délky k nazveme k-tici písmen $w = w_1 w_2 w_3 \cdots w_k, w_i \in \mathcal{A}$. Délku slova značíme $|w|$. Slovo nulové délky nazýváme prázdné slovo a značíme ϵ. Libovolnou posloupnost $\mathbf{d} = d_1 d_2 d_3 \cdots$, kde $d_i \in \mathcal{A}$, nazveme nekonečným slovem nad abecedou $\mathcal{A}$.

Množinu všech konečných slov označíme $\mathcal{A}^*$. Vybavíme-li množinu $\mathcal{A}^*$ binární asociativní operací zřetězení

$$u \cdot v = (u_1 u_2 \cdots u_m) \cdot (v_1 v_2 \cdots v_n) = u_1 u_2 \cdots u_m v_1 v_2 \cdots v_n = uv\,,$$

získáme monoid slov s neutrálním prvkem ϵ.

Konečné slovo w je faktorem slova $\mathbf{d}$, pokud existují slova (konečná nebo nekonečná) x a y taková, že $\mathbf{d} = xwy$.

Jazykem slova $\mathbf{d}$ nazveme množinu všech faktorů slova $\mathbf{d}$ a značíme

$$\mathcal{L}(\mathbf{d}) = \{w \big| w \text{ faktor } \mathbf{d}\}\,.$$

Množinu všech faktorů slova $\mathbf{d}$ délky m značíme

$$\mathcal{L}_m(\mathbf{d}) = \{w | w \in \mathcal{L}(\mathbf{d}), |w| = m\}\,.$$

Faktor u nazveme čtvercem, pokud $u = ww$, kde w je neprázdné slovo. Slovo $\mathbf{d}$ je square-free, pokud neobsahuje žádný čtverec. Je snadné nahlédnout, že nekonečné square-free slovo nemůže existovat nad unární ani binární abecedou.

Nekonečné slovo $\mathbf{d}$ je posléze periodické s periodou p, pokud je tvaru $\mathbf{d} = vw^\omega$, kde w^ω značí nekonečné opakování faktoru w. Pak $p = |w|$ volíme nejmenší možné. Pokud je předperioda $v = \epsilon$, nazveme slovo periodické. Slovo, které není posléze periodické nazýváme aperiodické.

Faktorovou komplexitou $\mathcal{C}_{\mathbf{d}} : \mathbb{N} \to \mathbb{N}$ slova $\mathbf{d}$ nazveme funkci přiřazující každému m počet faktorů slova $\mathbf{d}$ délky m:

$$\mathcal{C}_{\mathbf{d}}(m) = \#\mathcal{L}_m(\mathbf{d}).$$

Zobrazení $\tau : \mathcal{A}^* \to \mathcal{B}^*$ nazveme morfismem, pokud

$$\forall u, v \in \mathcal{A}^* \quad \tau(uv) = \tau(u)\tau(v)\,.$$

Působení morfismu můžeme přirozeným způsobem rozšířit i na nekonečná slova:

$$\tau(u_1u_2u_3\cdots) = \tau(u_1)\tau(u_2)\tau(u_3)\cdots$$

Morfismus τ je k-uniformní, jestliže

$$\forall a \in \mathcal{A} \quad |\tau(a)| = k\,.$$

Pokud nechceme zdůrazňovat přesnou délku obrazů písmen, pak říkáme, že τ je uniformní.

Slovo w je pevným bodem morfismu τ, pokud platí $w = \tau(w)$. Jestliže existuje takové písmeno $a \in \mathcal{A}$, že $\tau(a) = au, u \neq \epsilon$, a pro všechna $b \in \mathcal{A}$ platí $\lim_{n\to\infty} |\tau^n(a)| = +\infty$, pak nekonečné slovo $w = \tau^\infty(a)$ – tedy slovo w, jehož je $\tau^n(a)$ prefixem pro každé $n \in \mathbb{N}$ – je pevným bodem morfismu τ.

Morfismus τ nazveme square-free, pokud pro všechna square-free slova w je jejich obraz $\tau(w)$ také square-free.

Nyní uvedeme dvě věty, přičemž první poskytuje zachování square-free vlastnosti a dolní odhad pro komplexitu slov vzniklých proložením písmen. Druhá dává vztah mezi komplexitami slova a jeho morfického obrazu.

Věta 2. *Nechť $\mathbf{u} = u_1u_2u_3\cdots$ je slovo s komplexitou $\mathcal{C}_{\mathbf{u}}$ nad abecedou $\mathcal{A}$, dále $\mathbf{v} = v_1v_2v_3\cdots$ je square-free slovo nad abecedou $\mathcal{B}$ a $\mathcal{A} \cap \mathcal{B} = \emptyset$.*

Pak $\mathbf{u} = u_1v_1u_2v_2u_3v_3\cdots$ je square-free a platí $\mathcal{C}_{\mathbf{d}}(2m) \geq \mathcal{C}_{\mathbf{u}}(m)$.

Důkaz. Předpokládejme, že $\mathbf{u}$ obsahuje čtverec ww. Ten nemůže být liché délky, neboť pak by w začínalo písmenem z jedné abecedy a v druhém opakování z druhé, což je spor. Nechť tedy $|w|$ je sudá, pak buď na lichých nebo na sudých pozicích jsou písmena slova $\mathbf{v}$. Pak ale jsou tyto písmena samotné čtvercem, což je spor s předpokladem, že $\mathbf{v}$ je square-free.

Množina $\mathcal{L}_{2m}(\mathbf{d})$ obsahuje alespoň tolik faktorů jako $\mathcal{L}_m(\mathbf{u})$ kvůli písmenům z $\mathbf{u}$. □

Věta 3. *Nechť $\mathbf{u} = u_1u_2u_3\cdots$ je nekonečné slovo s komplexitou $\mathcal{C}_{\mathbf{u}}$ a ψ je k-uniformní prostý morfismus.*

Pak $\mathcal{C}_{\mathbf{u}}(m) \leq \mathcal{C}_{\psi(\mathbf{u})}(k \cdot m)$.

Důkaz. Z prostoty ψ musí být počet faktorů slova $\psi(\mathbf{u})$ délky km alespoň takový, jako počet faktorů slova $\mathbf{u}$ délky m. □

Pro konstrukci square-free slova s exponenciální komplexitou nad malou abecedou budeme potřebovat ještě následující větu.

Věta 4 (Brandenburg [6])**.** *Morfismus ψ prostý a uniformní je square-free právě tehdy, když $\psi(w)$ je square-free pro každé w square-free, $|w| = 3$.*

Důkaz. Pro lepší pochopení uvádíme i Brandenburgův důkaz.

Implikace zleva doprava je triviální.

Opačnou implikaci dokážeme sporem. Nechť tedy existuje square-free faktor $w, |w| \geq 4$ takový, že $\psi(w)$ obsahuje čtverec a w má minimální délku. Můžeme napsat $\psi(w) = xttz'$. Pak z minimality w existují písmena a, c taková, že $\psi(a) = xx'$ a $\psi(c) = zz'$ a $x' \neq \epsilon \neq z$ a $w = aw'c$. Jistě lze psát $w' = ubv$, kde b je písmeno, $\psi(b) = yy'$ a $t = x'\psi(u)y = y'\psi(v)z$. Rozlišíme dva případy.

1. Předpokládejme $|x'| = |y'|$. Přímo plyne $x' = y'$, z uniformity $\psi(u) = \psi(v)$ a $y = z$ a z prostoty $u = v$. Vezmeme-li $\psi(abc) = xx'yy'zz' = xx'yx'yz'$, pak abc není z předpokladu věty square-free, a tedy $a = b$ nebo $b = c$, čímž získáváme spor s předpokladem, že $w = aubvc = aubuc$ je square-free.

2. Nechť $|x'| > |y'|$, pak existuje $x'' \neq \epsilon$ takové, že $x' = y'x''$. Odtud $x''\psi(u)y = \psi(v)z$. Označíme $v = dv'$, kde d je písmeno. Zřejmě $|\psi(d)| > |x''|$ a $\psi(d) = x''d'$. Obraz $\psi(ad) = xx'x''d' = xy'x''x''d'$ obsahuje čtverec x'', a proto $a = d$. Z rovnosti $xy'x'' = \psi(a) = \psi(d) = x''d'$ víme, že $\psi(a) = x''\beta x'' = \psi(d), \beta \neq \epsilon$. Z $x''\psi(u)y = \psi(dv')z = x''\beta x''\psi(v')z$ plyne, že β je prefix $\psi(ub)$. Tím pádem $\psi(aub) = x''\beta x''\beta\cdots$ není square-free, což je spor s minimalitou w.

   Opačná nerovnost se ošetří obdobně.

□

5.2 Konstrukce ditheračních posloupností

V této sekci ukážeme, jak zkonstruovat nekonečná square-free slova s exponenciální komplexitou nad malými abecedami – velikosti čtyři, pět a šest – a nad velkou abecedou o 256 písmenech.

5.2.1 Čtyřpísmenná ditherační posloupnost

Pro zkonstruování slova nad čtyřpísmennou abecedou potřebujeme slovo $\mathbf{u}$ s exponenciální komplexitou a square-free slovo $\mathbf{v}$. Poté využijeme výše uvedených vět k jejich proložení a zmenšení abecedy.

Slovo $\mathbf{u}$ získáme řetězením všech možných binárních m-tic, tzn. binárních rozvojů čísel od 0 do $2^m - 1$ doplněných zleva nulami na délku m pro $m = 1, 2, 3, \ldots$:

$$\mathbf{u} = 0\,1\,00\,01\,10\,11\,000\,001\,010\,011\,100\,101\,110\,111\,0001\,0010\,0011\,0100\cdots . \tag{5.1}$$

Komplexita $\mathcal{C}_{\mathbf{u}}(m) = 2^m$, protože se z definice ve slově $\mathbf{u}$ vyskytují všechny faktory délky m.

Dále zkonstruujeme Thueovo square-free slovo $\mathbf{v}$. Pro konstrukci potřebujeme Thueovo-Morseovo slovo $\mathbf{u}_{TM}$ definované jako pevný bod morfismu $\nu^\infty(0)$, kde $\nu(0) = 01, \nu(1) = 10$:

$$\mathbf{u}_{TM} = 0110100110010110100101100110100\cdots .$$

Nyní definujeme písmena Thueova slova $\mathbf{u}_T$ jako počet jedniček mezi dvěma po sobě jdoucími nulami v Thueově-Morseově slově $\mathbf{u}_{TM}$:

$$\underbrace{\mathbf{u}_{TM}}_{\mathbf{u}_T} = 0\underbrace{11}_{2}0\underbrace{1}_{1}0\underbrace{}_{0}0\underbrace{11}_{2}0\underbrace{}_{0}0\underbrace{1}_{1}0\underbrace{11}_{2}0\ldots$$

Tedy

$$\mathbf{u}_T = 2102012101202102012021012102012\cdots .$$

Thueovo slovo $\mathbf{u}_T$ je square-free [23]. Potřebujeme jej nad jinou abecedou, než je abeceda slova $\mathbf{u}$, proto definujeme $\mathbf{v} = \eta(\mathbf{u}_T)$, kde $\eta(0) = 2, \eta(1) = 3, \eta(2) = 4$:

$$\mathbf{v} = \eta(\mathbf{u}_T) = 4324234323424324234243234324234\cdots . \tag{5.2}$$

Jinou možností je využít morfismu τ definovaného na abecedě $\{A, B, C, D\}$ jako

$$\tau(A) = AB, \quad \tau(B) = CA, \quad \tau(C) = CD, \quad \tau(D) = AC .$$

Tento morfismus má pevný bod

$$\mathbf{v}' = \tau^{\infty}(A) = ABCACDABCDACABCACDACABCDABCACDA\cdots , \tag{5.3}$$

na který aplikujeme morfismus

$$\mu(A) = 4, \quad \mu(B) = 3, \quad \mu(C) = 2, \quad \mu(D) = 3 .$$

Sporem lze ukázat, že získáme opět stejné slovo $\mu(\mathbf{v}') = \mathbf{v}$.

Nyní proložením $\mathbf{u}$ a $\mathbf{v}$ získáme slovo

$$\mathbf{d}_5 = 041302040213140312130402040302 1\cdots \tag{5.4}$$

nad abecedou {0,1,2,3,4}, které je podle věty 2 square-free s komplexitou $\mathcal{C}_{\mathbf{d}_5}(m) \geq 2^{\frac{m}{2}}$.

Brandeburg [6] uvádí příklad 18-uniformního prostého square-free morfismu ψ' z pětipísmenné abecedy do třípísmenné abecedy, což je nejkratší možný. Jako ditheračí posloupnost můžeme vzít

$$\mathbf{d}_3 = \psi'(\mathbf{d}_5) . \tag{5.5}$$

Podle věty 3 je komplexita $\mathcal{C}_{\mathbf{d}_3}(m) \geq 2^{\frac{m}{36}}$, tedy může být dost malá. Uvědomíme-li si, že pro reprezentaci tří písmen v počítači potřebujeme dva bity, můžeme rovnou využít čtyřpísmenné abecedy.

Nejkratší prostý square-free morfismus $\varphi : \{0, 1, 2, 3, 4\}^* \to \{0, 1, 2, 3\}^*$ je už jen 4-uniformní a může vypadat například takto:

$$\begin{aligned}
\varphi(0) &= 0102 ,\\
\varphi(1) &= 0123 ,\\
\varphi(2) &= 0312 ,\\
\varphi(3) &= 1013 ,\\
\varphi(4) &= 1032 .
\end{aligned}$$

Morfismus φ je nalezený podle věty 4 hrubou silou zkoušením prostých zobrazení přiřazujících jednotlivým písmenům square-free obrazy a následným ověřením, jestli jsou obrazy všech třípísmenných square-free slov nad abecedou {0,1,2,3,4} square-free.

Můžeme tedy pomocí φ sestrojit ditheračí posloupnost

$$\mathbf{d_4} = \varphi(\mathbf{d}_5) = 0102103201231013010203120102103\cdots , \tag{5.6}$$

která je nad čtyřpísmennou abecedou a má podle věty 3 komplexitu $\mathcal{C}_{\mathbf{d}_4}(m) \geq 2^{\frac{m}{8}}$.

Zdrojový kód programu pro hledání square-free morfismů `square_free_morphism.py` napsaný v Pythonu je na přiloženém CD.

5.2.2 Šestipísmenná ditherační posloupnost

Nebudeme-li chtít za každou cenu zmenšit abecedu ditherační posloupnosti, můžeme zkonstruovat nekonečné slovo s podstatně lepší komplexitou nad šestipísmennou abecedou následujícím způsobem – vezmeme opět slova $\mathbf{u}$ a $\mathbf{v}$ definované vztahy (5.1) a (5.2). Nebudeme je ovšem prokládat, ale definujeme zobrazení $\rho : \{0,1\} \times \{2,3,4\} \to \{a,b,c,d,e,f\}$ takto:

$$\begin{aligned}\rho(0,2) &= a\,,\\ \rho(0,3) &= b\,,\\ \rho(0,4) &= c\,,\\ \rho(1,2) &= d\,,\\ \rho(1,3) &= e\,,\\ \rho(1,4) &= f\,,\end{aligned}$$

neboli každou možnou dvojici označíme novým písmenem. Písmena slova $\mathbf{d}_6$ pak definujeme předpisem

$$d_{6,i} = \rho(u_i, v_i)\,. \tag{5.7}$$

Je zřejmé, že slovo

$$\mathbf{d}_6 = ceacaefbdecacbafaecafedbceafdec\cdots$$

je square-free s komplexitou $\mathcal{C}_{\mathbf{d}_6}(m) \geq 2^m$.

5.2.3 256písmenná ditherační posloupnost

Budeme-li písmena ditherační posloupnosti vkládat do vstupu kompresní funkce místo části zprávy, pak máme pro písmeno v podstatě jeden byte, neboť těžko budeme dělit zprávu na bloky jinak než po celých bytech. Máme tedy prostor pro dokonce 256písmennou abecedu.

Pro konstrukci slova využijeme slovo $\mathbf{u}$ s exponenciální komplexitou a čtyřpísmenné square-free slovo $\mathbf{v}'$ definovaná v (5.1) a (5.3). Písmeno posloupnosti $\mathbf{d}_{256}$ vytvoříme tak, že prvních šest bitů zaplníme šesti znaky slova $\mathbf{u}$, a zbylé dva bity využijeme pro zápis písmene slova $\mathbf{v}'$ ($\beta(A) = 00, \beta(B) = 01, \beta(C) = 10, \beta(D) = 11$). Tedy:

$$d_{256,i} = u_{6i-5}u_{6i-4}u_{6i-3}u_{6i-2}u_{6i-1}u_{6i}||\beta(v_i)\,. \tag{5.8}$$

Slovo $\mathbf{d}_{256}$ je zřejmě square-free díky písmenům ze slova $\mathbf{v}'$ a má komplexitu $\mathcal{C}_{\mathbf{d}_{256}}(m) \geq 2^{6m} = 64^m$, neboť slovo $\mathbf{u}$ obsahuje všechny faktory délky m, a tudíž bereme-li jeho šestice jako jedno písmeno, obsahuje všechny faktory délky m i toto slovo. Zdůvodnění je následující – šestice může vycházet přes řetězení dvou k-tic ve slově $\mathbf{u}$. Libovolnou sekvenci m šestic ale určitě najdeme ve slově $\mathbf{u}$ tam, kde vzniká řetězením všech možných $6 \cdot (m+1)$-tic.

5.3 Generování prefixů

Je-li nekonečné slovo konstruované jako pevný bod morfismu, jeví se jako nejpřirozenější způsob generování prefixu nekonečného slova opakování aplikace morfismu na počáteční písmeno, dokud nezískáme prefix požadované délky. Takto ale musíme pro získání n-tého písmene udržovat v paměti celý předchozí řetězec, paměťové požadavky jsou tedy $\mathcal{O}(n)$.

Algoritmus pro generování slova s paměťovou náročností pouze $\mathcal{O}(\ln n)$ uvedl J. Patera v článku [19]. Využívá substitučního stromu, ve kterém je v kořeni první písmeno nekonečného slova a potomci každého uzlu jsou označeni písmeny morfického obrazu písmena v uzlu v pořadí zleva doprava. Poté je strom procházen do šíře (tzn. po úrovních) zleva doprava s vynecháním levého podstromu, který představuje již nagenerovaný prefix. K průchodu do šíře je využíván zásobník, pomocí kterého si pamatujeme, které větve jsme již prošli. Díky stromové struktuře je potřeba pouze $\mathcal{O}(\ln n)$ paměti ke generování n-tého písmene.

Zdrojový kód (Python) generování prefixu slova $\mathbf{v}'$ vypadá následovně:

```
tau={'A':'AB','B':'CA','C':'CD','D':'AC'}
tauLen={}
for k in tau.keys():
    tauLen[k]=len(tau[k])
stack=[()]; stack.pop()
t0='A';  a=t0
n=1;     i=2
word=t0; N=1000
while (n<=N):
    while (i<=tauLen[a]):
        word+=tau[a][i-1]
        n+=1;  i+=1
    l=0
    while (len(stack)!=0 and i>tauLen[a]):
        (a,i)=stack.pop()
        l+=1;  i+=1
    if (len(stack)==0):
        a=t0;  i=2;  l+=1
    while 1:
        stack.append((a,i))
        a=tau[a][i-1]
        i=1;  l-=1
        if (l==0):
            break
print word
```

Algoritmus pro generování slova $\mathbf{u}$ vzniklého řetězením všech možných m-tic nad $\{0,1\}$ je následující:

```
m=1;  M=8;
word=""
while (m<=M):
    i=0
    while (i< (1<<m)):
        w=""
        for j in range(0,m):
            w=str((i&(1<<j))>>j)+w
        word+=w;  i+=1
    m+=1
print word
```

Pro ditherování potřebujeme v každé iteraci získat jen jedno písmeno. Toho lze docílit drobnou úpravou výše uvedených kódů. V souboru `words.py` jsou upravené zdrojové kódy společně s kódy pro generování prefixů dalších slov.

Každé slovo má svůj vlastní objekt. Rodičovským objektem všech dalších je objekt *Word*, který má metodu *print(n)* umožňující vytisknout prefix délky n. Všechny zděděné objekty mají metodu *first()* vracející první písmeno daného prefixu a metodu *next()* vracející při každém volání další písmeno posloupnosti. Tyto metody nám umožňují přidat do každého volání kompresní funkce jedno písmeno ditherační posloupnosti, aniž bychom museli generovat celý prefix naráz.

V objektu *WordMorphism* je implementován Paterův algoritmus. Jeho konstruktor přebírá jako parametr slovník popisující konkrétní morfismus τ ve tvaru

```
{'a_1':'tau(a_1)', ... , 'a_k':'tau(a_k)' }.
```

Od něj jsou odvozeny objekty *WordSqrFree4* a *WordThue*, které generují slova $\mathbf{v}'$ a $\mathbf{v}$.

Objekty *WordBinExp, Word4, Word5, Word6* a *Word256* generují slova $\mathbf{u}$, $\mathbf{d}_4$, $\mathbf{d}_5$, $\mathbf{d}_6$ a $\mathbf{d}_{256}$.

Kapitola 6

Ditherování

Kapitola se zabývá návrhem R. Rivesta [20] zlepšujícím odolnost hašovací funkce proti hledání druhého vzoru – ditherování. Využijeme poznatků z kombinatoriky na slovech k zavedení ditherování a zkoumání vlastností ditherované hašovací funkce. Popíšeme útoky na ditherované hašovací funkce a jejich složitosti v závislosti na volbě ditherační posloupnosti.

6.1 Konstrukce

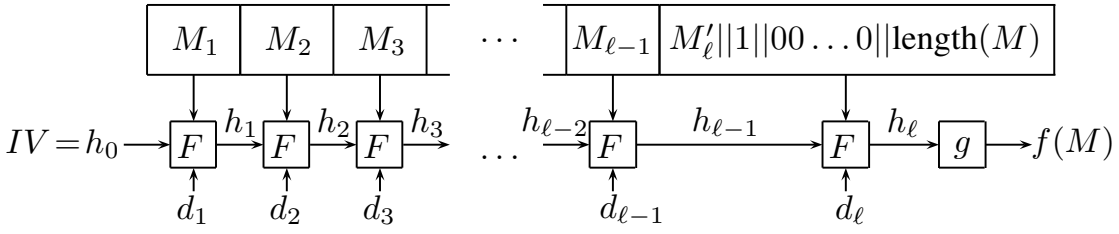
Při popisu útoku pomocí rozšiřitelné zprávy v sekci 4.1 jsme viděli, že iterativní konstrukce s sebou nese jisté vrozené vady, například neměnnost výsledné haše při řetězení pevných bodů. Nápadem R. Rivesta je přidat do každé iterace ještě jeden vstup (viz obrázek 6.1), tak aby průběžná haš závisela nejen na aktuálním bloku a předchozí haši, ale i na pozici bloku ve zprávě. Tuto úpravu nazval ditherování.[1]

Závislosti na pozici ve zprávě docílíme přidáním písmene d_i nekonečného slova $\mathbf{d}$ do vstupu kompresní funkce při každé iteraci:

$$h_i = F(h_{i-1}, M_i, d_i)\,.$$

Hašovací funkci ditherovanou slovem $\mathbf{d}$ označíme $f_{\mathbf{d}}$. V závislosti na zvolené ditherační posloupnosti se může zlepšit odolnost hašovací funkce vůči hledání druhého vzoru.

Vkládání ditheračního písmene si můžeme představit také tak, že máme k dispozici několik kompresních funkcí a ditherační písmeno určuje, kterou máme v dané iteraci použít.



Obrázek 6.1: Konstrukce ditherované hašovací funkce.

[1] Termín ditherování pochází z teorie počítačového zpracování obrazu, kde značí náhodné míchání pixelů při nedostatku odstínů nějaké barvy, aby se docílilo vizuálně plynulého přechodu (např. světlé oblohy do tmavé).

Uveďme nyní triviální příklady slova **d**:

- Konstantní posloupnost $\mathbf{d} = 00000\ldots$ odpovídá odolností klasické hašovací funkci.
- Periodická posloupnost, např.

  $$d_{2i-1} = 0\,, \qquad d_{2i} = 1\,,$$

  také odolnost příliš nezlepší, protože můžeme místo jednotlivých bloků opakovat p-tice bloků, kde p je délka periody slova **d**.
- Čítač, to znamená

  $$d_1 = 1, \qquad d_2 = 2, \qquad d_3 = 3, \ldots$$

  Problematická je narůstající velikost ditheračních písmen.

Vzhledem k výše uvedenému navrhuje R. Rivest použít k ditherování square-free slovo nad konečnou abecedou, které je nutně aperiodické, takže zabraňuje útoku pomocí rozšiřitelné zprávy. Faktory se ve square-free slově neopakují vůbec, což by mělo zabránit i útokům v podobě nějaké modifikace rozšiřitelné zprávy.

Modifikace útoku pomocí kolizního stromu popsaná v sekci 6.2 ukazuje, že je důležitá také komplexita a stejné frekvence výskytů faktorů ditherační posloupnosti.

Složitost útoků pomocí kite generátoru (sekce 6.3) a Hellmanových tabulek (sekce 6.4) zavisí pouze přímou úměrou na velikosti abecedy ditherační posloupnosti, ale poznamenejme, že složitost offline fáze těchto útoků je alespoň $\mathcal{O}(2^n)$. Útoky tedy nepředstavují reálnou hrozbu.

6.2 Útok pomocí kolizního stromu

Mějme hašovací funkci ditherovanou slovem **d** s kompresní funkcí $h_i = F(h_{i-1}, M_i, d_i)$. Značení je stejné jako v klasickém případě. Druhý vzor ke zprávě M_{target} nalezneme takto:

1. Najdeme slovo w délky $l+1$, které je faktorem slova **d** s nejvyšší frekvencí výskytu v prefixu **d** délky $2^k + l + 1$.
2. Zkonstruujeme kolizní strom hloubky l tak, že všechny hrany vycházející z listů ditherujeme symbolem w_1, do další úrovně w_2 a tak dále až do h_t vede hrana ditherovaná w_l (viz obrázek 6.2). Tvorba stojí $4\sqrt{2\ln 2}\cdot\sqrt{l}\cdot 2^{\frac{n+l}{2}}$ operací stejně jako pro klasické hašovací funkce, neboť kompresní funkce se chová jako náhodné orákulum, takže hledání kolizí hrubou silou není ditherováním nijak ovlivněno.
3. Hledáme blok M_{link} takový, že $F(h_t, M_{\text{link}}, w_{l+1}) = h_{i_0}$ pro nějaké $i_0 \in \mathcal{I}$,

   $$\mathcal{I} = \{i \in \mathbb{N} | (l+1 \leq i) \wedge (d_{i-l} \ldots d_i = w)\}\,.$$

   Frekvence výskytů faktoru w ve slově **d** je silně závislá na použité ditherační posloupnosti. Předpokládejme, že všechny faktory mají stejnou frekvenci výskytu, pravděpodobnost výskytu faktoru w ve slově **d** pak je $1/C_{\mathbf{d}}(l+1)$, přičemž $C_{\mathbf{d}}(n)$ je komplexita neboli počet faktorů délky n slova **d**. Odtud $\#\mathcal{I} \doteq 2^k/C_{\mathbf{d}}(l+1)$. Musíme tedy zkusit přibližně $C_{\mathbf{d}}(l+1)\cdot 2^{n-k}$ bloků.

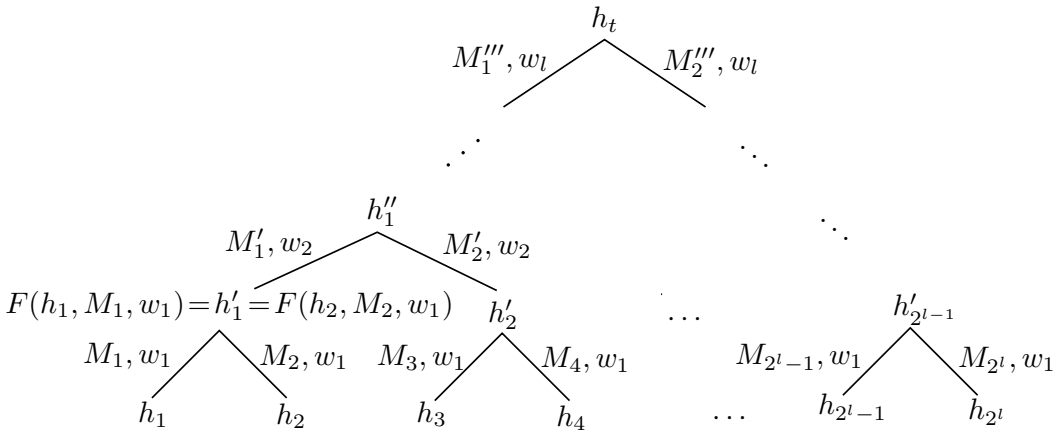
4. Pokud $i_0 = l + 1$, vezmeme za h'_{j_0} hodnotu IV. Jinak hledáme prefix P délky $i_0 - l - 1$, pro který platí $f_{\mathbf{d}}(IV, P) = h'_{j_0}$, $j_0 \in \widehat{2^l}$ (listy kolizního stromu). Potřebný počet volání kompresní funkce je 2^{n-l}.

5. Propojením stromu z h'_{j_0} do h_t vytvoříme zprávu T délky l.

6. Sestavíme druhý vzor M_{second}:

$$M_{\text{second}} := P||T||M_{\text{link}}||M_{i_0+1}||\dots||M_{2^k+l+1}\,.$$

Složitost celého algoritmu je $4\sqrt{2\ln 2}\cdot\sqrt{l}\cdot 2^{\frac{n+l}{2}} + C_{\mathbf{d}}(l+1)\cdot 2^{n-k} + 2^{n-l}$ volání kompresní funkce, závisí tedy na komplexitě ditherační posloupnosti $\mathbf{d}$.



Obrázek 6.2: Schéma kolizního stromu ditherovaného faktorem w.

6.3 Útok pomocí kite generátoru

Útok pomocí kite generátoru popsaný v článku [1] sice vyžaduje náročné předpočítání, autoři uvádějí $\mathcal{O}(|\mathcal{A}|\cdot 2^n)$, ale poté už umožňuje hledat druhé vzory mnohem menším počtem volání kompresní funkce, nezávisle na komplexitě ditherovací posloupnosti. Předpokládáme, že délka cílové zprávy je 2^k a ditherovací posloupnost $\mathbf{d}$ má abecedu $\mathcal{A}$. Aplikací podobných myšlenek jako v případě konstrukce kolizního stromu ukážeme, že složitost offline fáze je dokonce $\mathcal{O}(|\mathcal{A}|\cdot\sqrt{n-k}\cdot 2^n)$.

Základem útoku je vytvoření kite generátoru – struktury tvořené množinou K o 2^{n-k} různých haších, mezi kterými je i IV, a patřičnými bloky zpráv. Pro každou haš $h \in K$ a každé $\alpha \in \mathcal{A}$ najdeme dva bloky $m^{1,2}_{h,\alpha}$ takové, že $F(h, m^{1,2}_{h,\alpha}, \alpha)$ opět leží v množině K. Navíc ale požadujeme, aby každá haš v K byla obrazem dvou jiných haší pro každé α. Bloky $m^{1,2}_{h,\alpha}$ tedy musíme volit s ohledem na tento požadavek.

V online fázi nejprve vypočteme průběžné haše cílové zprávy $M = M_1 \cdots M_{2^k}$ a podle vztahu (2.3) nalezneme h_i ležící v K pro $i > n - k$. Nyní chceme pomocí kite generátoru vytvořit zprávu s výslednou haší $f(M)$:

1. Začneme v $IV = h'_0$ a pohybujeme se v kite generátoru tak, že bereme bloky $m_{h'_j,\alpha}$ podle ditherovací posloupnosti $\mathbf{d}$ až po $j = i - (n-k)$. Z možných dvou bloků si vždy můžeme zvolit libovolně.

2. Z bodu $h'_{i-(n-k)}$ rozvineme v kite generátoru podle příslušných písmen ditherovací posloupnosti binární strom obrazů hloubky $\frac{n-k}{2}$ pomocí bloků $m^{1,2}_{h'_j,\alpha}$. Množinu haší tvořících listy označíme L_{IV}.

3. Naopak z haše h_i rozvineme dle ditherovací posloupnosti binární strom vzorů kompresní funkce, opět hloubky $\frac{n-k}{2}$ a množinu listových haší označíme L_M. Grafickým znázorněním těchto dvou stromů vznikl název kite generátor – stromy jdoucí proti sobě připomínají papírového draka.

4. Množiny L_{IV} a L_M mají obě $2^{\frac{n-k}{2}}$ prvků z 2^{n-k} možných, a tedy mezi nimi nalezneme kolizní haš h'_c dle (2.3). Část zprávy vzniklé řetězením bloků značících cestu stromem z $h'_{i-(n-k)}$ do h'_c, respektive z h'_c do h_i označíme P_{IV}, resp. P_M.

5. Sestrojíme druhý vzor:

$$M_2 := m_{h'_0,d_1}||m_{h'_1,d_2}||\dots||m_{h'_{i-(n-k)},d_{i-(n-k)+1}}||P_{IV}||P_M||M_{i+1}||\dots||M_{2^k}\,.$$

6.3.1 Složitost konstrukce kite generátoru

Při počítání složitosti konstrukce kite generátoru opět využijeme poznatků z teorie náhodných grafů. Strukturu si podobně jako v případě kolizního stromu můžeme představit jako graf, jehož vrcholy jsou haše z libovolně zvolené množiny K a hranu představuje kolize kompresní funkce. Chceme najít dvě různá párování, výpočet tedy provedeme ve dvou krocích.

Podle sekce 2.2.1 musí pravěpodobnost p existence hrany, neboli kolize mezi dvěma hašemi, překročit prahovou funkci $\frac{\ln \nu}{\nu}$, kde ν je počet haší v K, tedy 2^{n-k}. Označíme-li počet vyzkoušených bloků ke každé haši L, platí podle (2.3) a požadavku, že kolizní haš má ležet v K:

$$p = \frac{L^2}{2^n} \cdot \frac{2^{n-k}}{2^n} = \frac{L^2}{2^n} \cdot \frac{1}{2^k}\,.$$

Porovnáním s prahovou funkcí získáváme

$$p = \frac{L^2}{2^{n+k}} = \frac{\ln 2^{n-k}}{2^{n-k}}\,.$$

Odtud

$$L = \sqrt{\ln 2}\sqrt{n-k} \cdot 2^k\,.$$

Musíme vyzkoušet L bloků pro každou haš v K a každé $\alpha \in \mathcal{A}$, počet volání kompresní funkce je proto

$$L \cdot |\mathcal{A}| \cdot 2^{n-k} = |\mathcal{A}| \cdot \sqrt{\ln 2}\sqrt{n-k} \cdot 2^k \cdot 2^{n-k}\,.$$

Nalezení druhého párování je analogické, pouze požadujeme, aby výsledné haše byly ty, které nemají vzory z prvního kroku, to znamená, že musí spadnout do zbylých $2^{n-k}/2$ haší, čímž se sníží p na polovinu, a tedy se v L objeví faktor $\sqrt{2}$. Celkový počet volání kompresní funkce při konstrukci kite generátoru je tedy alespoň

$$|\mathcal{A}| \cdot (1+\sqrt{2}) \cdot \sqrt{\ln 2}\sqrt{n-k} \cdot 2^n\,,$$

což je $\mathcal{O}(\sqrt{n-k} \cdot 2^n)$.

V uvažovaném postupu není vyloučeno, že některá z haší v generátoru bude obrazem více než dvou haší. Protože jsme ale hledali dvakrát párování v grafu, musela by tato haš být čtyř či více kolizí. Pro nalezení s-kolize musíme podle výsledků K. Suzukiho a spol. [22] provést $(s!)^{\frac{1}{s}} 2^{\frac{n(s-1)}{s}}$ hašování. Pro $s = 4$ je to $2^{1/2+3n/4}$ haší. Uvážíme-li, že máme v kite generátoru 2^{n-k} haší a $k > n/4$, bude tato čtyřkolize s velkou pravděpodobností nejvýše jedna, což můžeme zanedbat.

Paměťová náročnost je $\mathcal{O}(|\mathcal{A}| \cdot 2^{n-k})$.

6.3.2 Složitost online fáze útoku

Hledání h_i vyžaduje 2^k volání kompresní funkce pro výpočet průběžných haší. Rozvinutí stromů a hledání kolize mezi nimi vyžaduje $\mathcal{O}(2^{(n-k)/2})$ operací (nejedná se o volání kompresní funkce, pouze procházení kite generátoru a porovnávání listových haší) a paměti.

6.4 Útok pomocí Hellmanových tabulek

V článku [1] je popsán útok na ditherované hashovací funkce pomocí Hellmanových tabulek (viz sekce 2.4). Útok vyžaduje předpočítání se složitostí $\mathcal{O}(2^{n-k} + |\mathcal{A}| \cdot 2^n)$, ale poté umí hledat druhé vzory pro zprávy délky 2^k s časovou i paměťovou náročností $\mathcal{O}(2^{2(n-k)/3})$.

Hlavní myšlenkou je napojit předpočítané smyčky do cílové zprávy s odpovídajícími znaky ditherovací posloupnosti.

6.4.1 Offline fáze

Potřebujeme nalézt pro každé $\alpha \in \mathcal{A}$ blok zprávy X_α splňující $F(IV, X_\alpha, \alpha) = IV$. Jedinou možností je hledání hrubou silou, proto potřebujeme $|\mathcal{A}| \cdot 2^n$ volání kompresní funkce.

Dále si předpočítáme Hellmanovy tabulky pro každé $\alpha \in \mathcal{A}$ tak, aby pokrývaly 2^{n-k} možných haší, to znamená $m \cdot t \cdot d = 2^{n-k}$. Náhodnou funkci $f_\alpha : \mathsf{H} \to \mathsf{H}$ pro tvorbu tabulek příslušných k α definujeme jako $f_\alpha(h) := F(IV, h||0 \ldots 0, \alpha)$. Díky těmto tabulkám umíme nalézt blok $X := h||0 \ldots 0$ vedoucí z IV na haš h_i při ditherování písmenem α s pravděpodobností 2^{-k}.

6.4.2 Online fáze

Cílová zpráva tvořená bloky $M_1, \ldots, M_{2^k}$ má průběžné haše h_i. Ke každé z nich se postupně snažíme najít pomocí Hellmanových tabulek k patřičnému ditherovacímu písmenu blok zprávy X takový, že $F(IV, X, \alpha) = h_{i_0}$ pro nějaké $i_0 \in \widehat{2^k}$. Pravděpodobnost nalezení je 2^{-k} pro každý z 2^k bloků, měli bychom tedy jeden takový nalézt. Poté projíždíme smyčky pevných bodů tak, abychom dodrželi ditherovací posloupnost $\mathbf{d}$ a vytvořili prefix potřebné délky. Druhý vzor M_{second} pak sestavíme jako $X_{d_1}||X_{d_2}|| \ldots ||X_{d_{i_0-1}}||X||M_{i_0+1}|| \ldots ||M_{2^k}$.

Paměťová a výpočetní složitost online fáze je $\mathcal{O}(2^{2(n-k)/3})$, jak bylo vysvětleno v sekci 2.4.

6.5 Porovnání ditheračních slov

Nyní dosadíme do složitosti útoku pomocí kolizního stromu komplexity slov zkonstruovaných v sekci 5.2. Označme složitost útoku

$$S_{\mathbf{d}}(k,l,n) = \mathcal{O}\left(\sqrt{l}\cdot 2^{\frac{n+l}{2}} + C_{\mathbf{d}}(l+1)\cdot 2^{n-k} + 2^{n-l}\right).$$

Dosadíme-li ditherační posloupnost $\mathbf{d}_4$ danou vztahem (5.6) s komplexitou $\mathcal{C}_{\mathbf{d}_4}(m) \geq 2^{\frac{m}{8}}$, pak

$$S_{\mathbf{d}_4}(k,l,n) \geq \mathcal{O}\left(\sqrt{l}\cdot 2^{\frac{n+l}{2}} + 2^{\frac{l+1}{8}}\cdot 2^{n-k} + 2^{n-l}\right).$$

Pro slovo $\mathbf{d}_5$ ze vztahu (5.4), která má komplexitu $\mathcal{C}_{\mathbf{d}_5}(m) \geq 2^{\frac{m}{2}}$, je

$$S_{\mathbf{d}_5}(k,l,n) \geq \mathcal{O}\left(\sqrt{l}\cdot 2^{\frac{n+l}{2}} + 2^{\frac{l+1}{2}}\cdot 2^{n-k} + 2^{n-l}\right).$$

Další posloupností je $\mathbf{u}_6$ definovaná vzorcem (5.7) s komplexitou $\mathcal{C}_{\mathbf{d}_6}(m) \geq 2^m$. Pak

$$S_{\mathbf{d}_6}(k,l,n) \geq \mathcal{O}\left(\sqrt{l}\cdot 2^{\frac{n+l}{2}} + 2^{l+1}\cdot 2^{n-k} + 2^{n-l}\right).$$

Slovo $\mathbf{d}_{256}$ s největší komplexitou $\mathcal{C}_{\mathbf{d}_{256}}(m) \geq 2^{6m}$ je definované vztahem (5.8). Při ditherování tímto slovem je složitost útoku

$$S_{\mathbf{d}_{256}}(k,l,n) \geq \mathcal{O}\left(\sqrt{l}\cdot 2^{\frac{n+l}{2}} + 2^{6(l+1)}\cdot 2^{n-k} + 2^{n-l}\right).$$

Zvolíme-li $l = \frac{n}{3}$, tzn. tak, aby první a třetí člen měly stejné exponenty, a tudíž byly co nejmenší, jsou složitosti následující:

$$\begin{aligned}
S_{\mathbf{d}_4}(k,\frac{n}{3},n) &\geq \mathcal{O}\left(\sqrt{n}\cdot 2^{\frac{2n}{3}} + 2^{\frac{n}{24}}\cdot 2^{n-k}\right),\\
S_{\mathbf{d}_5}(k,\frac{n}{3},n) &\geq \mathcal{O}\left(\sqrt{n}\cdot 2^{\frac{2n}{3}} + 2^{\frac{n}{6}}\cdot 2^{n-k}\right),\\
S_{\mathbf{d}_6}(k,\frac{n}{3},n) &\geq \mathcal{O}\left(\sqrt{n}\cdot 2^{\frac{2n}{3}} + 2^{\frac{n}{3}}\cdot 2^{n-k}\right),\\
S_{\mathbf{d}_{256}}(k,\frac{n}{3},n) &\geq \mathcal{O}\left(\sqrt{n}\cdot 2^{\frac{2n}{3}} + 2^{2n}\cdot 2^{n-k}\right).
\end{aligned}$$

Připomeňme složitost útoku na klasickou hašovací funkci:

$$\begin{aligned}
S(k,l,n) &= \mathcal{O}\left(\sqrt{l}\cdot 2^{\frac{n+l}{2}} + 2^{n-k} + 2^{n-l}\right),\\
S(k,\frac{n}{3},n) &= \mathcal{O}\left(\sqrt{n}\cdot 2^{\frac{2n}{3}} + 2^{n-k}\right).
\end{aligned}$$

Pro minimalizaci složitosti musíme zvolit $k \geq \frac{n}{3}$. Při ditherování posloupnostmi $\mathbf{d}_4, \mathbf{d}_5$ a $\mathbf{d}_6$ musím volit k alespoň $\frac{3n}{8}$, $\frac{n}{2}$ a $\frac{2n}{3}$, prodlužuje se tedy minimální délka M_{target}, ke které umíme tímto útokem nalézt druhý vzor. Ditherování slovem $\mathbf{d}_{256}$ útoku pomocí kolizního stromu zcela zabrání (složitost je více než $\mathcal{O}(2^n)$).

Jak již bylo řečeno, složitosti offline fází útoku pomocí kite generátoru a Hellmanových tabulek se jen vynásobí velikostí abecedy příslušného ditheračního slova, z čehož vyplývá, že nejlepší je opět slovo $\mathbf{d}_{256}$.

Kapitola 7

Implementace ditherované hašovací funkce

V této kapitole krátce popíšeme, jakým způsobem lze využít kompresní funkci F klasické hašovací funkce k implementaci ditherované hašovací funkce. Ditherační písmeno můžeme xorovat na vstup kompresní funkce nebo jej vkládat místo části bloku zprávy. Ukážeme, že xorování ditherační posloupnosti na zprávu nijak nezvyšuje odolnost proti hledání druhého vzoru. Na závěr předvedeme konkrétní implementaci ditherované hašovací funkce využívající funkci MD5.

7.1 Xorování ditherančního písmene na vstup kompresní funkce

V článku [2] zkoumají J. Aumasson a R. Phan bezkoliznost ditherované hašovací funkce, xorujeme-li ditherační písmeno na jeden ze vstupů kompresní funkce.

První možností je xorovat ditherační písmeno na průběžné haše. Pak je kompresní funkce následující

$$F_{\mathbf{d}}^{\oplus h,1}(h_{i-1}, M_i, d_i) = F(h_{i-1} \oplus d_i, M_i)\,,$$

případně

$$F_{\mathbf{d}}^{\oplus h,2}(h_{i-1}, M_i, d_i) = F(h_{i-1}, M_i) \oplus d_i\,.$$

Proti této konstrukci sice nemáme konkrétní útok na druhý vzor, ale je zřejmé, že nám dává jistou možnost ovlivnit hodnotu průběžných haší.

Druhou možností je blok zprávy a ditherovací písmeno xorovat na sebe, kompresní funkce $F_{\mathbf{d}}^{\oplus}$ tedy vypadá takto:

$$F_{\mathbf{d}}^{\oplus}(h_{i-1}, M_i, d_i) = F(h_{i-1}, M_i \oplus d_i)\,.$$

Hašovací funkci s touto kompresní funkcí označme $f_{\mathbf{d}}^{\oplus}$. Jediné omezení autoři kladou na poslední blok zprávy, ve kterém se ditherovací posloupnost při xorování nesmí překrývat se samotnou zprávou, kvůli jednoduchému útoku na kolizi.

Budeme-li ale zkoumat odolnost vůči útoku na druhý vzor, zjistíme, že se zpráva s posloupností nesmí překrývat v žádném bloku, což odpovídá řetězení popsanému v sekci 7.2, jinak ditherování nijak nezvyšuje bezpečnost.

Útok na druhý vzor $f_{\mathbf{d}}^{\oplus}$ je jednoduchý:

1. K blokům dané M_{target} naxorujeme příslušnou ditherovací posloupnost $\mathbf{d}$:

$$M'_i = M^i_{\text{target}} \oplus d_i.$$

   Pro takto vytvořenou zprávu M' platí

$$f(M') = f^{\oplus}_{\mathbf{d}}(M_{\text{target}})\,,$$

   neboť do kompresní funkce vstupují tytéž bloky.

2. Ke zprávě M' najdeme druhý vzor M'_{second} bez použití ditherování, například pomocí rozšiřitelné zprávy. Tento útok zachovává poslední blok zprávy, takže dodrží omezení navržené autory.

3. Druhý vzor M_{second} k původní zprávě nalezneme opětovným naxorováním ditherovací posloupnosti na M'_{second}:

$$M^i_{\text{second}} = {M'}^i_{\text{second}} \oplus d_i.$$

   Opět platí

$$f(M'_{\text{second}}) = f^{\oplus}_{\mathbf{d}}(M_{\text{second}})\,.$$

   Protože M'_{second} je druhým vzorem M', platí

$$f^{\oplus}_{\mathbf{d}}(M_{\text{target}}) = f(M') = f(M'_{\text{second}}) = f^{\oplus}_{\mathbf{d}}(M_{\text{second}})\,.$$

Nalezli jsme tedy druhý vzor pro $f^{\oplus}_{\mathbf{d}}$ se stejnou složitostí jako bez použití ditherování.

7.2 Řetězení ditherační posloupnosti s blokem zprávy

Jinou možností, jak ditherovat hašovací funkci, je vkládat písmena ditherovací posloupnosti $\mathbf{d}$ místo části zprávy. Rozdělíme tedy zprávu na bloky M_i o délce vstupu klasické kompresní funkce F zkrácené o velikost písmene ditherovací abecedy. K těm pak připojíme ditherovací písmeno. Ditherovaná kompresní funkce $F_{\mathbf{d}}$ pak vypadá následovně:

$$F_{\mathbf{d}}(h_{i-1}, M_i, d_i) = F(h_{i-1}, M_i||d_i)\,.$$

Tímto způsobem můžeme implementovat ditherovanou hašovací funkci jednoduše tak, že zahašujeme klasickou funkcí vhodně překódovanou zprávu. Překódování provedeme tak, že na patřičná místa ve zprávě vložíme ditherační písmena. Doplnění zprávy jedničkou, nulami a délkou zprávy je třeba provést tak, aby po doplnění všech ditheračních písmen byla délka zprávy násobkem délky vstupu kompresní funkce.

7.3 Implementace ditherovaní do funkce MD5

Pro ilustraci je na přiloženém CD v souboru `md5_dithered.py` uvedena hašovací funkce využívající zdrojový kód MD5 [21] ditherovaná slovem $\mathbf{d}$. Ditherování probíhá tak, že se v každé iteraci bere kratší blok zprávy a doplňuje se ditheračním písmenem.

Pro ditherační písmeno se využívá celý byte a kóduje se tak, že se jeho zápis přečte jako hexadecimálně zapsané přirozené číslo.

Na obrázku 7.1 jsou haše několika různých řetězců bez ditherování a při ditherování slovy $\mathbf{d}_4$, $\mathbf{d}_5$, $\mathbf{d}_6$ a $\mathbf{d}_{256}$.

Zdrojový kód je v jazyce Python, navíc bez zvláštních optimalizací, takže není vhodný pro testování rychlosti ditherované hašovací funkce, ale je výrazně jednodušší a přehlednější než například v jazyce C, v němž by hašovaní bylo podstatně rychlejší.

Funkce *md5_dithered(message,dithSeq)* bere jako vstup zprávu *message* k hašování ve formě bitového řetězce a objekt typu *Word* jako ditherační posloupnost *dithSeq*. Vrací hodnotu haše jako číslo typu *Long*, které můžeme převést do hexadecimálního zápisu funkcí *md5_to_hex(digest)*.

```
Message: ""
MD5:     d41d8cd98f00b204e9800998ecf8427e
Word4:   d41d8cd98f00b204e9800998ecf8427e
Word5:   d41d8cd98f00b204e9800998ecf8427e
Word6:   aae938d287b57ec51ec383bad75329e6
Word256: 1ec0bd70e69e191d1008d5df2958ec15

Message: "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
MD5:     d174ab98d277d9f5a5611c2c9f419d9f
Word4:   dc3d3c14d6a95eeee129bd8763f81296
Word5:   d8a76961ccb99b020bf50ec6efb7c6bb
Word6:   86deebdf251c17b1df11a38db179e7f1
Word256: 54c35123d20f67e4684d00f8cd5d47de

Message: "The quick brown fox jumps over the lazy dog"
MD5:     9e107d9d372bb6826bd81d3542a419d6
Word4:   014bc2dee8142c4c5962cebab80ff581
Word5:   014bc2dee8142c4c5962cebab80ff581
Word6:   b9a75b97c36218bdc4f3f543ce222fa7
Word256: 6bd7c3b565399ab2e9ce746aacc38d49

Message: "The quick brown fox jumps over the lazy dog."
MD5:     e4d909c290d0fb1ca068ffaddf22cbd0
Word4:   c13d1e64acd741e5bc011fe4aff39069
Word5:   c13d1e64acd741e5bc011fe4aff39069
Word6:   8bc4fb12f07c0ec5bcad14bdefc7bc81
Word256: c22eeaa8f17fe1c5a18dbe2961f69773

Message: "A hash function is any algorithm that maps data of variable
 length to data of a fixed length. The values returned by a hash
 function are called hash values, hash codes, hash sums, checksums
 or simply hashes."
MD5:     562e009eace597fb6b9690b864ea869f
Word4:   2784e269b94e1f0e1d054ba3e2a75316
Word5:   f0d04b6f50fee83fea870a33a483776c
Word6:   7567beb2639b318d59e3ba160e0c9f09
Word256: dd2915b2cebabccb5e132658e7711ee7
```

Obrázek 7.1: Srovnání klasické MD5 a ditherované slovy $\mathbf{d}_4$, $\mathbf{d}_5$, $\mathbf{d}_6$ a $\mathbf{d}_{256}$

Kapitola 8

Další konstrukce

V roce 2007 byla americkým úřadem pro standardizaci (NIST) vyhlášena soutěž o nový standard SHA-3 (Secure Hash Algorithm) jako reakce na prolomení funkce MD5, možné slabiny funkce SHA-1 a nedostatečný výkon SHA-2. V prosinci 2010 bylo vyhlášeno pět finalistů – BLAKE, Grøstl, JH, Keccak a Skein. Bohužel do finále nepostoupil algoritmus BMW, jehož spoluautorem je český kryptolog Vlastimil Klíma, přestože předčil ostatní svojí rychlostí. Zdůvodnění výběru finalistů je ve zprávě NISTu [18]. Vítězem byl v říjnu 2012 vyhlášen algoritmus Keccak, který využívá tzv. *sponge* funkci.

V této kapitole krátce popíšeme konstrukce HAIFA a wide-pipe, které byly využity při návrhu některých algoritmů do soutěže SHA-3.

V sekci 8.3 naopak předvedeme hledání multikolizí zobecnění LAB módu navrženého X. Yao, které navdory prvnímu dojmu nepřináší výrazný bezpečnostní přínos.

8.1 Wide-pipe

Jako obranu proti hledání multikolizí, ale také dalším útokům, navrhl S. Lucks [15] používat wide-pipe neboli dvoj- či vícenásobnou pumpu. Jedná se v podstatě o řešení hrubou silou – zvětšíme (zdvojnásobíme) délku průběžných haší, čímž zabráníme hledání kolizí narozeninovým útokem a tím i všem dalším generickým útokům. Výsledná haš se potom získá zkrácením poslední průběžné na požadovanou délku.

Zprávu zarovnáme stejně jako v MD zesílení. Následně iterujeme kompresní funkci $F_{\text{wide}} : \{0,1\}^{k \cdot n} \times \mathsf{B} \to \{0,1\}^{k \cdot n}$ postupně pro bloky $M_1, M_2, \ldots, M_\ell$:

$$h_i = F_{\text{wide}}(h_{i-1}, M_i)\,.$$

Na výslednou haš poté aplikujeme funkci $F_{\text{cut}} : \{0,1\}^{k \cdot n} \to \mathsf{H}$:

$$f(M) = F_{\text{cut}}(h_\ell)\,.$$

Pro zamezení Jouxovu útoku na multikolize je postačující volit $k = 2$.

Tento koncept využívá například algoritmus Grøstl a BMW.

8.2 HAIFA

Jako obranu proti útokům plynoucím z iterativní konstrukce publikovali E. Biham a O. Dunkelman v článku [4] HAsh Iterative FrAmework – HAIFA.[1] Konstrukce je podobná ditherování, v každé iteraci totiž přidává počet již zahašovaných bitů a částečně upravuje Merkleovo-Damgårdovo zesílení v posledním bloku.

Haš spočteme takto – zprávu doplníme jedničkou, nulami, délkou zprávy v bitech a délkou otisku na násobek velikosti bloku. Délka otisku se přidává, protože HAIFA umožňuje produkovat otisky různé délky a tímto zabráníme jejich kolizi:

$$M = M_1 M_2 \dots M_{\ell-1} \underbrace{M'_\ell ||1||00\dots0||\text{length}(M)||n}_{m \text{ bitů}} .$$

Poté ℓ-krát aplikujeme kompresní funkci $C : \mathsf{H} \times \mathsf{B} \times \{0,1\}^b \times \{0,1\}^s \to \mathsf{H}$. Průběžné haše se počítají jako

$$\begin{aligned} h_0 &= IV \,, \\ h_i &= C(h_{i-1}, M_i, \#bits, salt) \,, \end{aligned}$$

kde $\#bits$ je počet již zahašovaných bitů a $salt$ je sůl, což je hodnota, která může být zvolena při každém výpočtu haše jinak. Výslednou haší je hodnota h_ℓ. Sůl zvyšuje bezpečnost hašovací funkce v aplikacích, kde ji útočník nezná předem, a tudíž zabrání předpočítání.

Pokud chceme haš délky $l < n$, použijeme jako inicializační vektor hodnotu

$$IV_l = C(l, IV, 0, 0) \,,$$

a z výstupu funkce vezmeme prvních l bitů.

Konstrukce HAIFA je odolná vůči útokům na druhý vzor ze stejných důvodů jako hašovací funkce ditherovaná slovem s velkou komplexitou a navíc brání předpočítání, pokud je možné utajit před útočníkem hodnotu soli.

Z konceptu HAIFA vychází algoritmus BLAKE a Skein.

8.3 LAB mód

V preprintu [24] navrhuje X. Yao používat jako ditherovací posloupnost předchozí blok zprávy, případně sumu předchozích bloků, a tvrdí, že takto lze vylepšit odolnost hašovací funkce nejen proti útoku na druhý vzor, ale také proti Jouxovu útoku na multikolizi až na teoretickou hodnotu. Tento mód nazývá Locking Abutting Blocks, tedy zamykání přiléhajících bloků.

Zobecníme jeho konstrukci a ukážeme, že pro určitou třídu funkcí, která zahrnuje oba Yaovy návrhy, je její odolnost proti hledání multikolizí srovnatelná s klasickou Merkleovou-Damgårdovou konstrukcí. Navíc je výpočet haše při této konstrukci minimálně dvakrát pomalejší.

Ukážeme tedy, že s ohledem na hledání multikolizí nemá opodstatněný význam zakomponovávat do kompresní funkce předcházející bloky.

[1] Inspirace pro název je zřejmá – autoři žijí v izraelském městě Haifa.

8.3.1 Základní LAB mód

Rozdělíme zprávu na ℓ bloků M_j o poloviční délce vstupu kompresní funkce. Jako vstup poté použijeme vždy dva po sobě jdoucí bloky:

$$h_i = F_{LAB}(h_{i-1}, M_{i-1}, M_i) = F(h_{i-1}, M_{i-1}||M_i)\,,$$

kde M_0 je pevně daná konstanta a h_ℓ použijeme jako výstup hašovací funkce f_{LAB}.

8.3.2 Vylepšený LAB mód

Vylepšená verze LAB módu zapracovává pomocí sumy do výpočtu průběžné haše všechny předcházející bloky:

$$\sum^{j} M = M_1 \oplus M_2 \oplus \cdots \oplus M_j\,,$$

případně i průběžné haše:

$$\sum^{j} M = M_1 \oplus M_2 \oplus \cdots \oplus M_j \oplus h_0 \oplus h_1 \oplus \cdots \oplus h_{j-1}\,.$$

Rozdělíme tedy zprávu na ℓ bloků M_j o poloviční délce vstupu kompresní funkce a jako vstup poté použijeme aktuální blok a průběžnou sumu:

$$h_i = F'_{LAB}(h_{i-1}, \sum^{i-1} M, M_i)\,,$$

kde M_0 je pevně daná konstanta. Hodnotu h_ℓ použijeme jako výstup hašovací funkce f'_{LAB}.

8.3.3 Zobecněná konstrukce

Naše zobecnění spočívá v přidání libovolné funkce předcházejících bloků a průběžných haší do každé iterace. Nechť je $g_i : \mathsf{B}^i \times \mathsf{H}^i \to \mathsf{B}$ pro $i \geq 2$ funkcí bloků $M_1, \ldots, M_i$ a k nim příslušných průběžných haší $h_0, \ldots, h_{i-1}$. Dále $g_0, g_1 \in \mathsf{B}$ jsou libovolně pevně zvolené konstanty. Pak je iterace následující:

$$h_i = \widetilde{F}(h_{i-1}, g_{i-1}(M_1, \ldots, M_{i-1}, h_0, \ldots, h_{i-2}), M_i)\,.$$

Výstupem hašovací funkce $\widetilde{f}$ je hodnota h_ℓ.

Každá průběžná haš tedy závisí na všech předchozích. To ale zpomaluje výpočet haše. Navíc pokud nebudeme chtít uchovávat v paměti všechny předchozí bloky, pak se g_i bude napočítávat také iterativně, a tudíž bude podléhat útoku uvedenému dále (za předpokladu, že g_i nebude jednocestná, což by ale opět zvýšilo složitost výpočtu haše).

Je zřejmé, že pro základní LAB mód je

$$g_i(M_1, \ldots, M_i, h_0, \ldots, h_{i-1}) = M_i\,,$$

a pro vylepšený LAB mód

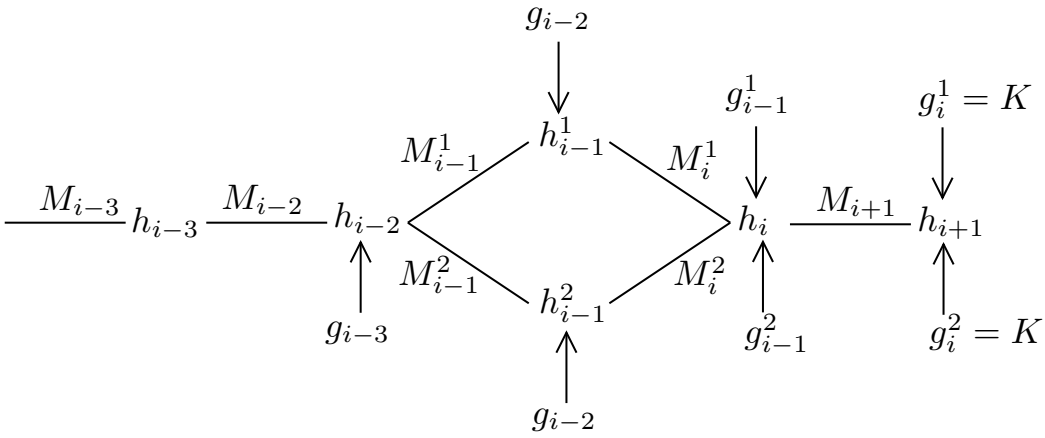
$$\begin{aligned} g'_i(M_1, \ldots, M_i, h_0, \ldots, h_{i-1}) &= M_1 \oplus M_2 \oplus \cdots \oplus M_i = \\ &= g'_{i-1}(M_1, \ldots, M_{i-1}, h_0, \ldots, h_{i-2}) \oplus M_i\,, \end{aligned}$$

případně i s průběžnými hašemi:

$$\begin{aligned} g''_i(M_1, \ldots, M_i, h_0, \ldots, h_{i-1}) &= M_1 \oplus M_2 \oplus \cdots \oplus M_i \oplus h_0 \oplus h_1 \oplus \cdots \oplus h_{i-1} \\ &= g''_{i-1}(M_1, \ldots, M_{i-1}, h_0, \ldots, h_{i-2}) \oplus M_i \oplus h_{i-1}\,. \end{aligned}$$

8.3.4 Hledání multikolizí zobecněné konstrukce

Nyní budeme po funkcích g_i požadovat jisté vlastnosti, které nám umožní provést jednoduchou analogii Jouxova útoku (viz sekce 3.2). Myšlenkou je využití dvou bloků k nalezení kolize průběžné haše, přičemž druhý blok upraví hodnotu funkce g_i tak, aby při hašování dalších bloků nezávisela na zvolení bloků z kolize, viz obrázek 8.1.



Obrázek 8.1: Schéma hledání kolizní dvojice bloků.

Nechť je $\forall i \geq 2$ funkce g_i taková, že

$$\forall M_1, \ldots, M_{i-2} \in \mathsf{B}\, \exists K \in \mathsf{B}\, \forall M_{i-1}\, \exists M_i\ (g_i(M_1, \ldots, M_i, h_0, \ldots, h_{i-1}) = K)\ .$$

Počet operací potřebných k nalezení M_i označme s. Vlastnost tedy popisuje jistou invertibilitu funkce g_i v i-té proměnné.

Dále požadujeme, aby platilo $\forall M_1, \ldots, M_j\, \forall i, j \in \mathbb{N}, i < j$:

$$\begin{aligned}
&(\exists M_{i-1}^1, M_i^1, M_{i-1}^2, M_i^2 \in \mathsf{B}, M_{i-1}^1 \neq M_{i-1}^2, M_i^1 \neq M_i^2 \\
&(g_i(M_1, \ldots, M_{i-2}, M_{i-1}^1, M_i^1, h_0, \ldots, h_{i-1}) = g_i(M_1, \ldots, M_{i-2}, M_{i-1}^2, M_i^2, h_0, \ldots, h_{i-1}))) \implies \\
&\implies g_j(M_1, \ldots, M_{i-2}, M_{i-1}^1, M_i^1, M_{i+1}, \ldots M_j, h_0, \ldots, h_{j-1}) = \\
&\quad = g_j(M_1, \ldots, M_{i-2}, M_{i-1}^2, M_i^2, M_{i+1}, \ldots M_j, h_0, \ldots, h_{j-1})\,,
\end{aligned}$$

což můžeme shrnout tak, že pokud se rovnají hodnoty g_i pro dvě různé dvojice po sobě jdoucích bloků, pak jsou stejné i hodnoty g_j pro tyto dvě dvojice pro $j > i$.

Poznamenejme, že pokud se g_i počítá rekurentně jako

$$g_i(M_1, \ldots, M_i, h_0, \ldots, h_{i-1}) = G(g_{i-1}(M_1, \ldots, M_{i-1}, h_0, \ldots, h_{i-2}), M_i, h_{i-1})\,,$$

je druhá vlastnost automaticky splněna.

Mějme tedy zobecněnou konstrukci popsanou výše se systémem funkcí $g_i, i \in \mathbb{N}$ s požadovanými vlastnostmi. Popíšeme nyní, jak k začátku zprávy $M_{\text{begin}} = M_1 M_2 \ldots M_{i-2}$ a konci zprávy $M_{\text{end}} = M_{i+1} M_{i+2} \ldots M_L$ nalézt dvě dvojice bloků M_{i-1}^1, M_i^1 a M_{i-1}^2, M_i^2 takové, že $\widetilde{f}(M_{\text{begin}}||M_{i-1}^1||M_i^1||M_{\text{end}}) = \widetilde{f}(M_{\text{begin}}||M_{i-1}^2||M_i^2||M_{\text{end}})$.

Algoritmus je následující:

1. Zvolíme $2^{\frac{n}{2}}$ různých bloků M_{i-1}.
2. Za s operací najdeme ke každému bloku M_{i-1} podle první vlastnosti blok M_i, tak aby
$$g_i(M_1, \ldots, M_i, h_0, \ldots, h_{i-1}) = K\,.$$
3. Pro dvojice M_{i-1}, M_i spočteme haše
$$h_i = \widetilde{F}\left(h_{i-1}, g_{i-1}(M_1, \ldots, M_{i-1}, h_0, \ldots, h_{i-2}), M_i\right) =$$
$$= \widetilde{F}\left(\widetilde{F}\left(h_{i-2}, g_{i-2}(M_1, \ldots, M_{i-2}, h_0, \ldots, h_{i-3}), M_{i-1}\right), g_{i-1}(M_1, \ldots, M_{i-1}, h_0, \ldots, h_{i-2}), M_i\right).$$
4. Mezi získanými hašemi najdeme podle narozeninového paradoxu kolizi. Kolizní dvojice označíme M^1_{i-1}, M^1_i a M^2_{i-1}, M^2_i.

Je zřejmé, že
$$\begin{aligned} h^1_{i-1} &= \widetilde{F}(h_{i-2}, g_{i-2}(M_1, \ldots, M_{i-2}, h_0, \ldots, h_{i-3}), M^1_{i-1}) \neq \\ &\neq \widetilde{F}(h_{i-2}, g_{i-2}(M_1, \ldots, M_{i-2}, h_0, \ldots, h_{i-3}), M^2_{i-1}) = h^2_{i-1}\,, \end{aligned}$$
což ale ničemu nevadí. Podstatné je, že
$$g_i(M_1, \ldots, M_{i-2}, M^1_{i-1}, M^1_i, h_0, \ldots, h_{i-1}) = g_i(M_1, \ldots, M_{i-2}, M^2_{i-1}, M^2_i, h_0, \ldots, h_{i-1}) = K\,,$$
a proto
$$\begin{aligned} h^1_{i+1} &= \widetilde{F}(h^1_i, g_i(M_1, \ldots, M_{i-2}, M^1_{i-1}, M^1_i, h_0, \ldots, h_i), M_{i+1}) = \\ &= \widetilde{F}(h_i, K, M_{i+1}) = \\ &= \widetilde{F}(h^2_i, g_i(M_1, \ldots, M_{i-2}, M^2_{i-1}, M^2_i, h_0, \ldots, h_i), M_{i+1}) = h^2_{i+1}\,. \end{aligned}$$
Díky druhé vlastnosti jsou shodné i všechny následující haše a tedy
$$\widetilde{f}(M_{\text{begin}}||M^1_{i-1}||M^1_i||M_{\text{end}}) = \widetilde{f}(M_{\text{begin}}||M^2_{i-1}||M^2_i||M_{\text{end}})$$

Pokud nalezneme takové kolizní dvojice bloků na k různých místech, získáme 2^k-kolizi za $k \cdot 2 \cdot 2^{\frac{n}{2}}$ volání kompresní funkce a $k \cdot s \cdot 2^{\frac{n}{2}}$ operací potřebných k invertování funkce g_i.

Z algoritmu útoku na druhý vzor vidíme, že můžeme zeslabit požadavky na funkce g_i tak, že u obou vlastností budeme místo $\forall i$ požadovat pouze platnost pro $i_1, i_2, \ldots, i_k$ takové, že $i_1 < i_2 < \cdots < i_k$ a $i_{j+1} - i_j \geq 2, j \in \widehat{k-1}$. Těchto k pozic nám totiž stačí k vytvoření 2^k-kolize.

Ukažme ještě, jak nalézt blok M_i pro základní LAB mód:
$$M_i = K\,.$$
Pro vylepšený LAB mód:
$$M_i = K \oplus \sum^{i-2} M \oplus M_{i-1}\,,$$
případně
$$M_i = K \oplus \sum^{i-2} M \oplus h_{i-2} \oplus M_{i-1} \oplus h_{i-1}\,.$$
Odtud je vidět, že potřebný počet operací s je nějaká nízká konstanta, a proto můžeme nalézt 2^k-kolizi všech verzí LAB módu se složitostí $\mathcal{O}(k \cdot 2^{\frac{n}{2}})$, tedy stejnou jako pro klasickou Merkleovu-Damgårdovu konstrukci, nikoli požadovanou teoretickou složitostí, jak tvrdí X. Yao.

8.3.5 Tvorba kolizního stromu

Analogickým způsobem jako při tvorbě kolize – tedy využitím dvou bloků tak, že druhý blok „opraví“ hodnotu g_i – můžeme sestrojit i kolizní strom.

Konkrétně pro základní mód budou hranu ve stromě místo jednoho bloku tvořit dva bloky s tím, že druhý bude stejný pro celé patro. Takto se složitost tvorby kolizního stromu oproti klasické konstrukci hašovací funkce pouze zdvojnásobí.

Při napojování kolizního stromu do zprávy M_{target}, ke které chceme nalézt druhý vzor, je postup následující – napojujeme pomocí dvou bloků $M_{1,2}$. M_2 volíme takový, který se v M_{target} vyskytuje nejčastěji. Vyskytuje-li se ve zprávě 2^l-krát, napojíme strom do M_{target} vyzkoušením 2^{n-l} bloků M_1. Složitost tedy velice silně závisí na zprávě M_{target}, na rozdíl od ditherování, u kterého je pro všechny M_{target} stejná. Patologický případ, kdy jsou všechny bloky M_{target} totožné, má složitost stejnou jako klasická konstrukce hašovací funkce, což se při ditherování vhodnou posloupností nestane.

Prefix připojovaný do kolizního stromu pro získání zprávy správné délky musí končit stejným blokem, který byl použit jako pevný při konstrukci listového patra kolizního stromu.

Pro zobecněnou konstrukci můžeme kolizní strom bez obtíží zkonstruovat a stejně tak připojit prefix, ale napojování kolizního stromu do M_{target} už silně závisí na konkrétní podobě funkcí g_i.

Shrnutí

Cílem této bakalářské práce bylo seznámit se s hašovacími funkcemi a útoky na ně, dále pak s ditherováním a k tomu nezbytnou teorií z kombinatoriky na slovech. Navrhli jsme vhodné ditherační posloupnosti včetně zdrojových kódů umožňujících iterativně generovat jejich prefixy. Ditherování těmito slovy jsme jako příklad implementovali do funkce MD5.

Podařilo se dosáhnout i drobných nových výsledků:

1. Oprava složitosti konstrukce kite generátoru na $\mathcal{O}(|\mathcal{A}| \cdot \sqrt{n-k} \cdot 2^n)$ místo $\mathcal{O}(|\mathcal{A}| \cdot 2^n)$.
2. Útok na nalezení druhého vzoru na ditherovanou hašovací funkci, jejíž kompresní funkce je konstruována xorováním ditherančního písmene na blok zprávy. Ukázali jsme, že ditherování tímto způsobem nepřináší žádné vylepšení vlastností.
3. Útok na multikolize LAB módu a jeho zobecnění. Vyvrátili jsme tím tvrzení X. Yao, že tento mód dosahuje požadované teoretické složitosti.

Shrnující článek o hašovacích funkcích a jejich ditherování je pod stejným názvem jako bakalářská práce odeslán do *Pokroků matematiky, fyziky a astronomie*. Spolupracujeme také s X. Yao na krátkém článku (note), která popisuje zanedbatelný přínos přidávání předchozích bloků do kompresní funkce.

V budoucnu bychom se chtěli věnovat zpřesnění složitosti útoků, co se týče pravděpodobnosti jejich úspěchu, a celkově preciznějšímu pojetí bezpečnosti v kryptografii.

Použité zdroje

[1] E. Andreeva, Ch. Bouillaguet, P.-A. Fouque, J. J. Hoch, J. Kelsey, A. Shamir, and S. Zimmer. Second Preimage Attacks on Dithered Hash Functions. *Advances in Cryptology – EUROCRYPT 2008, Lecture Notes in Computer Science 4965*, pages 270–288, 2008.

[2] J. P. Aumasson and R. C.-W. Phan. How (Not) to Efficiently Dither Blockcipher-Based Hash Functions? In *Progress in Cryptology – AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 308–324. Springer Berlin Heidelberg, 2008.

[3] L. Balková. Nahlédnutí pod pokličku kombinatoriky na nekonečných slovech. *Pokroky matematiky, fyziky a astronomie*, 56(1):9–18, 2011.

[4] E. Biham and O. Dunkelman. A Framework for Iterative Hash Functions - HAIFA. *IACR Cryptology ePrint Archive*, page 278, 2007.

[5] S. R. Blackburn, D. R. Stinson, and J. Upadhyay. On the Complexity of the Herding Attack and some Related Attacks on Hash Functions. *Designs, Codes and Cryptography*, 64(1-2):171–193, 2012.

[6] F. J. Brandenburg. Uniformly Growing k-th Power-Free Homomorphisms. *Theoretical Computer Science*, 23:69–82, 1983.

[7] I. B. Damgård. A Design Principle for Hash Functions. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 416–427, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[8] H. Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11(4):253–271, 1998.

[9] P. Erdős and A. Rényi. On the Evolution of Random Graphs. In *Publication of the Institute of Mathematics of the Hungarian Academy of Science*, pages 17–61, 1960.

[10] P. Erdős and A. Rényi. On the Existence of a Factor of Degree One in a Connected Random Graph. In *Acta Math. Acad. Sci. Hungar*, pages 359–368, 1966.

[11] M. E. Hellman. A Cryptoanalytic Time–Memory Trade-Off. *IEEE Transactions on Information Theory*, IT-26:401–406, 1980.

[12] A. Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology-Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

[13] J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.

[14] J. Kelsey and B. Schneier. Second Preimages on n-bit Hash Functions for Much Less than 2^n Work. In *Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 474–490, Berlin, Heidelberg, 2005. Springer-Verlag.

[15] S. Lucks. Design Principles for Iterated Hash Functions. *IACR Cryptology ePrint Archive*, page 253, 2004.

[16] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[17] R. C. Merkle. A Certified Digital Signature. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 218–238. Springer-Verlag New York, Inc., 1989.

[18] Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition. `http://csrc.nist.gov/publications/nistir/ir7764/nistir-7764.pdf`.

[19] J. Patera. Generating the Fibonacci Chain in $\mathcal{O}(\log n)$ Space and $\mathcal{O}(n)$ Time. *Phys. Part. Nuclei*, 33:225–234, 2002.

[20] R. L. Rivest. Abelian Square-free Dithering for Iterated Hash Functions. *Presented at ECrypt Hash Function Workshop, June 21, 2005, Cracow, and at the Cryptographic Hash workshop, November 1, 2005, Gaithersburg, Maryland (August 2005)*, 2005.

[21] MD5 Implementation in Python. `http://rosettacode.org/wiki/MD5/Implementation#Python`. Ze dne: 19. 6. 2012.

[22] K. Suzuki, D. Tonien, K. Kurosawa, and K. Toyota. Birthday Paradox for Multi-collisions. In *Information Security and Cryptology – ICISC 2006*, volume 4296 of *Lecture Notes in Computer Science*, pages 29–40. Springer Berlin Heidelberg, 2006.

[23] A. Thue. Über Unendliche Zeichenreihen. In *Norske Vid. Selsk. Skr.*, pages 1–22. I. Mat. Nat. Kl., Christiana 7, 1906.

[24] X. Yao. LAB Form for Iterated Hash Functions. *IACR Cryptology ePrint Archive*, page 269, 2010.