Hašovací funkce a kombinatorika na slovech

(Hash Functions and Combinatorics on Words)

Ľubomíra Balková, Jan Legerský, Praha

Abstrakt

Hašovací funkce jsou „zázračné“ funkce, které z dlouhých zpráv vyrábějí jejich krátké otisky (anglicky *hash*). Chovají se jako tzv. náhodná orákula, tedy otisky jsou jakoby náhodně losovány. Dále musí být funkce jednocestné, tj. musí splňovat, že hašování, tedy získávání otisku zprávy, je rychlé, zatímco k danému otisku je výpočetně nemožné najít zprávu, která by měla tento otisk. Podle konkrétní aplikace pak klademe na hašovací funkce ještě další požadavky.

Ukazuje se, že iterativní princip, na kterém je většina současných hašovacích funkcí založena, není dostatečně silný proti útoku na druhý vzor a multikolize. Ronald Rivest navrhl jako jedno z řešení metodu ditherování. V článku se zabýváme studiem jejích výhod a nevýhod a využíváme výsledků z kombinatoriky na slovech pro konstrukci vhodných ditheračních posloupností.

Na závěr se ještě krátce zmíníme o výsledcích soutěže o nový hašovací standard SHA-3, kterým se 2. října 2012 stal algoritmus Keccak.

1 Hašovací funkce

Hašovací funkce $f : \{0,1\}^N \to \{0,1\}^n$ je zobrazení přiřazující zprávě M délky maximálně N řetězec pevně dané délky n, kde $N >> n$, a má následující vlastnosti [11]:

- Je snadné spočítat haš (otisk) zprávy $f(M)$ (čas $\mathcal{O}(N)$, paměť $\mathcal{O}(1)$).
- Je odolná vůči nalezení vzoru: k dané haši h_{target} je výpočetně nemožné nalézt zprávu M takovou, že $f(M) = h_{\text{target}}$.
- Je odolná vůči nalezení druhého vzoru: k dané zprávě M_{target} je výpočetně nemožné nalézt jinou zprávu M takovou, že $f(M) = f(M_{\text{target}})$.
- Je odolná vůči kolizi: je výpočetně nemožné nalézt různé zprávy M a M' takové, že $f(M) = f(M')$.
- Chová se jako náhodné orákulum: při jakékoliv změně vstupu se na výstupu změní každý bit s pravděpodobností 50%.

Z vlastností hašovací funkce je zřejmé, že není prostá a že se velké množství zpráv zobrazí na stejnou haš (průměrně 2^{N-n}). Přesto ale požadujeme, aby bylo výpočetně nemožné najít vzor, druhý vzor i kolizi. Pod slovíčkem „výpočetně nemožné“ se skrývají dvě věci:

1. Dnešní počítače nesmí umět takovou úlohu řešit v reálném čase.
2. Složitost nalezení vzoru, druhého vzoru i kolize musí být blízká složitosti teoretické, přičemž – jak vysvětlíme v druhé kapitole – teoretická složitost nalezení vzoru i druhého vzoru je úměrná 2^n volání hašovací funkce, zatímco složitost nalezení kolize je úměrná $2^{\frac{n}{2}}$ volání hašovací funkce. Konstanty úměrnosti jsou dané pravděpodobností, s jakou chceme vzor, druhý vzor či kolizi najít.

Poznamenejme ještě, že všechny vlastnosti požadované po hašovací funkci jsou důležité, žádná neplyne z jiné (ačkoliv to možná čtenář na první pohled nevidí). Sice například platí, že jakmile najdeme druhý vzor, našli jsme vlastně i kolizi. Ovšem

my se zajímáme o to, zda je složitost jejich nalezení blízká teoretické složitosti, a ty jsou pro kolizi a druhý vzor různé.

Pokud by bylo jednoduché najít druhý vzor, může dojít i k vážnému zneužití. Digitální podepisování chápe haš jako jednoznačnou identifikaci dokumentu, podobně jako je jedinečný otisk prstu. Právě haš zprávy se podepisuje. Pokud by bylo možné najít k haši podepsané zprávy smysluplný druhý vzor, pak by digitální podpis nebyl spolehlivý.

****
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: V roce 1996 H. Dobbertin prezentoval na konferenci FSE 1996 metodu nalézání kolizí u algoritmu MD4.

Pokud funkce není bezkolizní, není považována za kryptograficky bezpečnou. Na obrázku 1 jsou dva dokumenty se stejnou haší (to je umožněno zdánlivě bezvýznamnými znaky v textu reprezentovanými hvězdičkami). Jeden z nich by mohl být podstrčen k podpisu a druhý (falešný) dokument by pak také měl stejný podpis. Snadné nacházení kolizí svědčí o některých slabinách algoritmu.

1.1 Narozeninový paradox

Ilustrujme rozdíl mezi hledáním druhého vzoru a kolize na známém narozeninovém paradoxu. Odvodíme vzorce pro pravděpodobnosti nalezení kolize a druhého vzoru. A z nich určíme složitost nalezení kolize a druhého vzoru s pravděpodobností alespoň 50%.

Začněme například kolizí. Mějme množinu o m prvcích. Vybíráme k prvků po jednom s vracením. Pravděpodobnost kolize, tedy pravděpodobnost, že je ve výběru některý prvek alespoň dvakrát:

$$P_1(m,k) = 1 - \frac{m(m-1)\dots(m-k+1)}{m^k}.$$

Snadno nahlédneme, že pro m velké a $k \doteq (2m \cdot \ln 2)^{1/2}$ je $P_1(m,k) \doteq 50\%$. Odtud pro počet prvků m roven počtu možných haší 2^n a počet volání hašovací funkce zhruba $(2\ln 2)^{1/2} \cdot 2^{\frac{n}{2}}$ dostáváme zhruba 50procentní pravděpodobnost nalezení kolize. Pravděpodobnost nalezení druhého vzoru, tedy pravděpodobnost, že najdeme mezi vybranými prvky předem daný prvek:

$$P_2(m,k) = 1 - \frac{(m-1)^k}{m^k}.$$

Odtud vidíme, že pro m velké a $k = m \cdot \ln 2 \doteq m$ je $P_2(m,k) \doteq 50\%$. Odtud pro počet prvků m roven počtu možných haší 2^n a počet volání hašovací funkce cca $\ln 2 \cdot 2^n$ dostáváme zhruba 50procentní pravděpodobnost nalezení druhého vzoru.

Ze vzorců také odvodíme známá fakta, že pro narozeninovou party, kde mají mít s alespoň 50procentní pravděpodobností dva hosté narozeniny ve stejný, ale libovolný den, stačí, aby se party účastnilo 23 lidí. Platí totiž $P_1(365, 23) \doteq 0,507$.

Ovšem pokud jde někdo na party a chce, aby tam s 50procentní pravděpodobností měl některý z hostů narozeniny ve stejný den jako on, pak je třeba 253 hostů. Samozřejmě předpokládáme rovnoměrné rozdělení narozenin během roku (tj. ignorujeme přestupné roky, dvojčata atd.)

2 Použití hašovacích funkcí

Hašovací funkce se využívají na mnoha místech. Podle užití se liší nároky, které na ně klademe: největší požadavky na bezkoliznost a odolnost vůči hledání druhého vzoru jsou kladeny v kryptografii, v jiných oblastech je spíše hlavní rychlost hašování. Vyjmenujme alespoň několik použití: autentizace dat, digitální podpisy, ověřování a ukládání hesel, identifikace dat nebo souborů, hašovací tabulky, generátory pseudonáhodných čísel, prokazování znalosti nebo autorství.

2.1 Digitální podepisování

Podepisovací algoritmus je časově náročná procedura. Proto se zpráva nejprve zahašuje a teprve haš zprávy se podepisuje. Z velkých dat vytváří hašovací funkce jejich identifikátor (říkáme také digitální otisk (anglicky *digital fingerprint*) nebo výtah zprávy (anglicky *message digest*)). V řadě zemí stojí digitální otisky na úrovni otisků prstů pro identifikaci lidí, proto při digitálním podpisu zprávy stačí podepsat její haš. Díky odolnosti vůči hledání druhého vzoru nemůže pak být taková zpráva zfalšována, tedy nalezena jiná zpráva se stejnou haší, a tedy stejným podpisem.

2.2 Autentizace uživatele

Hašování se používá v elektronické komunikaci k ověření totožnosti uživatele. Odesílatel k otevřené zprávě přiloží ještě text zprávy zahašovaný spolu s klíčem, který sdílí s adresátem. Ten po přijetí zprávu také zahašuje se sdíleným klíčem a ověří totožnost haše od odesílatele s tou, kterou vytvořil. Pokud by útočník změnil text zprávy, haše by se neshodovaly (klíčem totiž útočník nedisponuje), a tím by byla manipulace zprávy odhalena.

Pokud se jedná o pouhé ověření totožnosti uživatele bez posílání zprávy, dotazovatel (server) odešle uživateli výzvu (anglicky *challenge*), kterou uživatel opět spolu se sdíleným klíčem zahašuje a haš odešle zpět.

2.3 Shodnost dat

Dalším využitím hašování je kontrola kopírovaných dat, jde tedy o ochranu před náhodnými chybami. K tomu se využívají speciální hašovací funkce, na které nejsou kladeny zvláštní bezpečnostní nároky. Takovou hašovací funkcí je např. CRC (*Cyclic Redundancy Check*). Funkce vytváří kontrolní součet, který je vzhledem k velikosti souboru miniaturní, ale již při nepatrném rozdílu v souboru se změní. Srovnává se tedy kontrolní součet původního souboru a přeneseného. V případě shody jsou téměř jistě soubory identické, v případě neshody jednoznačně odlišné.

2.4 Ověřování a ukládání hesel

Místo hesel se ukládají jejich haše. Když dojde k nabourání systému, nejsou hesla díky jednocestnosti hašovací funkce kompromitována. Během přihlašování do systému se přenáší po síti pouze vypočtená hodnota haše, která se porovnává s uloženou hodnotou.

3 Konstrukce hašovacích funkcí

Zatím jsme se dozvěděli, jaké neskromné nároky klademe na hašovací funkce. Je tedy přirozené se ptát jak takové „zázračné“ funkce konstruovat. 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 [5, 12].

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

1. Rozsekáme zprávu M na bloky velikosti m:

   $$M_1 M_2 \dots M_{\ell-1} M'_\ell || 1 || 00 \dots 0 || \text{length}(M)\,.$$

   Do posledního bloku jsme doplnili jedničku a nuly tak, aby posledních t bitů bylo vyhrazeno na binární zápis délky zprávy. Takové úpravě se říká Merkleovo-Damgårdovo zesílení.

2. Iterujeme ℓ-krát *kompresní funkci* $F(h_{i-1}, M_i) : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ 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.

A v čem spočívá výhoda Merkleova-Damgårdova schématu? Autoři dokázali, že je-li použita jejich konstrukce, pak odolnost kompresní funkce F proti kolizím implikuje odolnost celé hašovací funkci f proti kolizím. No a hlídat odolnost kompresní funkce proti kolizím je daleko jednodušší, protože taková funkce zkracuje bloky délky $n+m$ na bloky délky n, zatímco hašovací funkce komprimuje zprávy délky až $N = 2^t - 1$ na haše délky n. Pro konkrétní představu uveďme, že při použití hašovací funkce SHA512 je $t = 128$, $m = 1024$ a $n = 512$ bitů.

3.1 Slabiny iterativní konstrukce

Dosud nejčastěji užívané hašovací funkce MD5, SHA-1 a SHA-2 využívají Merkleovo-Damgårdovo paradigma a také Daviesovu-Meyerovu konstrukci, která je založena na aplikaci blokové šifry $E_k(x)$. Kompresní funkce pak vzniká xorováním (sčítáním bit po bitu modulo 2) blokové šifry s průběžnou haší, kde klíčem blokové šifry je blok zprávy $k = M$ a „šifruje se“ průběžná haš:

$$F(h, M) = E_M(h) \oplus h\,.$$

Slabinou Daviesovy-Meyerovy konstrukce je snadné hledání pevných bodů. Při známém klíči je totiž bloková šifra invertovatelná (aby příjemce, který zná klíč, mohl bez problémů dešifrovat). To znamená, že ke každé zprávě M umíme najít právě jeden kontext h tak, že $h = F(h, M)$:

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

Na základě této slabiny nalezli R. D. Dean [6] a J. Kelsey a B. Schneier [8] útok na druhý vzor pomocí rozšiřitelné zprávy (anglicky *expandable message*), což je struktura umožňující produkovat zprávy délky v daném rozmezí se stejnou poslední průběžnou haší. Tato konstrukce nezahrnuje Merkleovo-Damgårdovo zesílení, proto nevznikají přímo kolize zpráv, ale rozšiřitelnou zprávu můžeme použít k hledání druhého vzoru napojením na určité místo cílové zprávy. Popišme konstrukci rozšiřitelné zprávy a útok na druhý vzor pomocí rozšiřitelné zprávy podrobněji.

Mějme zprávu M_{target} délky 2^k bloků, tedy $M_{target} = M_1 M_2 \dots M_{2^k}$.

1. Najdeme $2^{\frac{n}{2}}$ pevných bodů (h_i, M_{fix}), tzn. $h_i = F(h_i, M_{fix})$, volením různých bloků M_{fix}.
2. Vypočteme $2^{\frac{n}{2}}$ haší $h'_j = f(IV, M'_1)$ libovolným vybíráním M'_1.
3. Mezi těmito dvěma seznamy haší najdeme kolizi díky narozeninovému paradoxu. Společnou haš označíme h_{exp}.
4. Vytvoříme rozšiřitelnou zprávu $M(\ell)$ (počet bloků ℓ si volíme libovolně):

   $$M(\ell) = M'_1 || M_{fix}^{\ell-1} .$$

   Platí

   $$f(IV, M(\ell)) = h_{exp} .$$

5. Hledáme zprávu M_{link} takovou, že $F(h_{exp}, M_{link}) = \widetilde{h}_j$ pro nějaké $j \in \{2, \dots, 2^k\}$, kde $\widetilde{h}_j$ jsou průběžné haše M_{target}.
6. Hledaným druhým vzorem zprávy M_{target} je pak zpráva

   $$M'_1 || M_{fix}^{j-1} || M_{j+1} \dots M_{2^k} .$$

Složitost tohoto útoku na druhý vzor je $2^{\frac{n}{2}+1} + 2^{n-k}$ volání kompresní funkce, což je pro dlouhé zprávy (tedy velká k) mnohem méně než požadovaná složitost blízká 2^n.

R. Rivest [13] navrhl jako obranu proti útoku pomocí rozšiřitelné zprávy metodu *ditherování*, která souvisí s kombinatorikou na slovech.

4 Kombinatorika na slovech

Zrodu kombinatoriky na slovech byl v Pokrocích matematiky, fyziky a astronomie věnován článek [2]. Pro úplnost ale připomeneme všechny potřebné pojmy z kombinatoriky na slovech i zde.

Abecedou $\mathcal{A}$ rozumíme konečnou množinu symbolů, říkáme jim *písmena*. *Slovem* w nazýváme konečnou posloupnost písmen. Jeho *délkou* rozumíme počet písmen, která obsahuje, a značíme $|w|$. $\mathcal{A}^*$ značíme množinu všech konečných slov nad abecedou $\mathcal{A}$ s přidáním prázdného slova ε (jde o neutrální prvek vůči operaci řetězení). Množina $\mathcal{A}^*$ vybavená operací řetězení tvoří monoid. Pod *nekonečným slovem* $\mathbf{u}$ nad abecedou $\mathcal{A}$ pak rozumíme nekonečnou posloupnost písmen z $\mathcal{A}$, tj. $\mathbf{u} = u_1 u_2 u_3 \cdots$, kde $u_i \in \mathcal{A}$. Konečné slovo w nazveme *faktorem* (*podslovem*) konečného či nekonečného slova u, pokud existuje slovo v a slovo t (konečné či nekonečné) tak, že $u = vwt$. Je-li v prázdné slovo, nazveme w *prefixem* slova u. *Jazykem* $\mathcal{L}(\mathbf{u})$ nekonečného slova $\mathbf{u}$ nazýváme množinu všech faktorů tohoto slova. Nechť n je přirozené číslo, symbolem $\mathcal{L}_n(\mathbf{u})$ značíme množinu všech faktorů délky n slova $\mathbf{u}$. Často pracujeme

s morfismy φ: $\mathcal{A}^* \to \mathcal{B}^*$, kde $\mathcal{A}$, $\mathcal{B}$ jsou abecedy. (Morfismus je samozřejmě jednoznačně dán, pokud známe obrazy písmen abecedy $\mathcal{A}$.) Jejich působení lze přirozenou cestou rozšířit i na nekonečná slova:

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

Pokud nekonečné slovo $\mathbf{u}$ splňuje $\varphi(\mathbf{u}) = \mathbf{u}$, nazveme jej *pevným bodem* morfismu φ. Morfismus nazýváme *n-uniformní*, pokud obrazy všech písmen mají délku n.

Faktorovou komplexitou slova $\mathbf{u}$ nazveme funkci $\mathcal{C} : \mathbb{N} \to \mathbb{N}$, která každému n přiřadí počet faktorů délky n slova $\mathbf{u}$, tj.

$$\mathcal{C}(n) = \#\mathcal{L}_n(\mathbf{u}).$$

Slovo $\mathbf{u}$ je *bez čtverců* (anglicky *square-free*), pokud neobsahuje žádný faktor ve tvaru ww, kde w je libovolný neprázdný faktor slova $\mathbf{u}$. O morfismu φ řekneme, že je *bez čtverců* (*square-free*), pokud pro každé slovo w bez čtverců platí, že obraz $\varphi(w)$ je také bez čtverců.

Příklad 1. *Ilustrujme uvedené pojmy na nekonečném slově $\mathbf{u} = (abb)^\omega$, kde ω značí nekonečné opakování. Abeceda je $\mathcal{A} = \{a, b\}$. Slovo babb je faktorem délky 4 slova $\mathbf{u}$. Slovo abbabba je prefixem délky 7 slova $\mathbf{u}$. Snadno nahlédneme, že množina všech faktorů délky 5 slova $\mathbf{u}$ je $\mathcal{L}_5(\mathbf{u}) = \{abbab, bbabb, babba\}$. Dokonce pro každou délku $n \geq 2$ existují právě 3 faktory délky n, proto $\mathcal{C}(n) = 3$ pro $n \geq 2$. Definujeme-li morfismus $\varphi(a) = abb$ a $\varphi(b) = abb$, pak je zřejmě $\mathbf{u}$ pevným bodem φ. Slovo $\mathbf{u}$ obsahuje čtverce, protože např. abbabb je faktorem $\mathbf{u}$.*

5 Ditherování

Název ditherování je vypůjčen ze zpracování obrazu, kde znamená simulaci barev, které nemáme, náhodným mícháním pixelů podobných barev s cílem odstranit nežádoucí linie. R. Rivest navrhl následující metodu [13], kterou nazval ditherováním, k vylepšení odolnosti hašovací funkce proti útoku na druhý vzor.

Při každé iteraci přidáváme do vstupu kompresní funkce písmeno d_i nekonečného slova $\mathbf{d}$:

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

Je-li nekonečné slovo $\mathbf{d}$ bez čtverců, pak ditherování znemožní útok pomocí rozšiřitelné zprávy. Ten byl totiž založen na faktu, že jsme snadno uměli hledat pevné body kompresní funkce, tj. bloky M splňující $h = F(h, M)$, které pak při řetězení za sebe produkovaly stále stejnou průběžnou haš. Nyní ovšem, máme-li pevný bod, tj. $h = F(h, M, d)$, nepomůže nám řetězit bloky M za sebe. Dostáváme sice stejné průběžné haše, ovšem při použití ditherační posloupnosti d^k. Takové slovo ale jistě není faktorem $\mathbf{d}$, proto nelze zopakovaný blok M snadno použít pro tvorbu rozšiřitelné zprávy. Fakt, že $\mathbf{d}$ neobsahuje čtverce, nám dává i obranu proti útoku pomocí „skoro“ pevných bodů, kdy umíme hledat blok zprávy M_i tak, že $h_{i-k} = F(h_{i-1}, M_i, d_i)$ pro nějaké k.

Jak už to v kryptologii bývá, přišli J. Kelsey, A. Shamir *et al.* s útokem na ditherovanou hašovací funkci [1]. Zobecnili známý útok pomocí tzv. kolizního stromu i pro ditherované hašovací funkce, přičemž se ukázalo, že složitost útoku se zvětší úměrně komplexitě použitého ditheračního slova. Jde tedy o úspěšný útok proti konkrétním Rivestovým návrhům ditherovaných hašovacích funkcí, ve kterých pracoval s ditheračním slovem majícím lineární komplexitu.

Pokud ovšem zvolíme ditherační slovo s vysokou komplexitou (takové konstrukce popíšeme v následující kapitole) nebo velkou abecedou (např. ditherování čítačem v hašovací funkci HAIFA [3]), znemožníme útok pomocí kolizního stromu a znesnadníme i některé další útoky.

5.1 Ditherované hašovací funkce

V této kapitole se pokusíme využít znalosti z kombinatoriky na slovech tak, abychom získali nekonečná slova vhodná pro ditherování hašovacích funkcí. Cílem je konstruovat slova, která budou bez čtverců a jejich komplexita bude růst exponenciálně.

Věta 1. *Nechť* $\mathbf{u} = u_1u_2u_3\cdots$ *je nekonečné slovo s komplexitou* $\mathcal{C}_\mathbf{u}(n)$ *nad abecedou* $\mathcal{A}$*, dále* $\mathbf{v} = v_1v_2v_3\cdots$ *je nekonečné slovo bez čtverců nad abecedou* $\mathcal{B}$ *a* $\mathcal{A}\cap\mathcal{B} = \emptyset$*. Pak* $\mathbf{d} = u_1v_1u_2v_2u_3v_3\cdots$ *je bez čtverců a pro jeho komplexitu platí* $\mathcal{C}_\mathbf{d}(2n) \geq \mathcal{C}_\mathbf{u}(n)$*.*

Příklad 2. *Nekonečné slovo* $\mathbf{u}$ *s exponenciální komplexitou* $\mathcal{C}_\mathbf{u}(n) = 2^n$ *získáme například řetězením binárních rozvojů přirozených čísel:*

$$\mathbf{u} = 11011100101110111\cdots .$$

Nekonečné slovo $\mathbf{v}$ *neobsahující čtverce získáme následovně:*
Uvažujme morfismus τ *definovaný na abecedě* $\{A, B, C, D\}$ *jako*

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

Aplikujeme-li opakovaně morfismus τ *na písmeno* A*, pak získáváme delší a delší prefixy* $A, \tau(A), \tau^2(A), \ldots$ *nekonečného slova, které označíme* $\tau^\infty(A)$ *a které je pevným bodem morfismu* τ*. Aplikujeme-li posléze na* $\tau^\infty(A)$ *morfismus*

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

pak získané slovo $\mathbf{v} = \mu(\tau^\infty(A)) = 432423432342\cdots$ *je nekonečné slovo bez čtverců.*

Proložením $\mathbf{u}$ *a* $\mathbf{v}$ *dostaneme nekonečné slovo bez čtverců*

$$\mathbf{d} = 14130214121304031203141 21\cdots$$

nad abecedou $\{0, 1, 2, 3, 4\}$ *s komplexitou* $\mathcal{C}_\mathbf{d}(n) \geq 2^{\frac{n}{2}}$*.*

Poznámka 1. *Výše uvedená konstrukce slova bez čtverců je rychlejší, ovšem méně intuitivní než původní konstrukce, kterou navrhl A. Thue a jež byla popsána v Pokrocích [2]. Připomeňme i onu původní konstrukci.*

Definujme morfismus φ_{TM} *na abecedě* $\{0, 1\}$ *jako*

$$\varphi_{\mathrm{TM}}(0) = 01 \quad a \quad \varphi_{\mathrm{TM}}(1) = 10.$$

Pak Thueovo-Morseovo slovo $\mathbf{u}_{\mathrm{TM}}$ *je pevným bodem tohoto morfismu začínajícím písmenem* 0*.*

A. Thue dále zkonstruoval nekonečné slovo $\mathbf{v}$ *nad abecedou* $\{0, 1, 2\}$*, které neobsahuje čtverce. Pro* $n \geq 1$ *označil jako* v_n *počet jedniček mezi* n*-tým a* $(n+1)$*-vým výskytem nuly v Thueově-Morseově slově. Hledané slovo pak získal jako* $\mathbf{v} = v_1v_2v_3\cdots$*. Tedy*

$$\underbrace{\mathbf{u}_{\mathrm{TM}}}_{\mathbf{v}} = 0\underbrace{11}_{2}0\underbrace{1}_{1}0\underbrace{}_{0}0\underbrace{11}_{2}0\underbrace{}_{0}0\underbrace{1}_{1}0\underbrace{11}_{2}0\cdots .$$

Pokud přeznačíme $0 \to 2$*,* $1 \to 3$ *a* $2 \to 4$*, pak si čtenář s trochou píle může ověřit, že* $\mathbf{v}$ *je rovno slovu zkonstruovanému výše jako* $\mu(\tau^\infty(A))$*.*

5.2 Velikost abecedy

Při konstrukci ditheračního slova vypadá přirozeně požadavek, aby jeho abeceda byla co nejmenší a písmena ditheračního slova tak zabírala minimální místo ve vstupu kompresní funkce. F.-J. Brandenburg [4] ukázal, že existuje morfismus bez čtverců z abecedy libovolné velikosti do třípísmenné abedcedy. Vezmeme-li jeho příklad $\psi : \{0,1,2,3,4\}^* \to \{0,1,2\}^*$, můžeme získat nekonečné slovo $\mathbf{d}'$ bez čtverců s exponenciální komplexitou $\mathcal{C}_{\mathbf{d}'}(n) \geq 2^{\frac{n}{36}}$ nad abecedou {0,1,2} jako

$$\mathbf{d}' = \psi(\mathbf{d})\,.$$

Morfismus ψ je 18-uniformní, což je nejkratší možný.

Návodem pro hledání morfismů bez čtverců je následující věta:

Věta 2 (Brandenburg)**.** *Morfismus ψ injektivní a uniformní je bez čtverců právě tehdy, když $\psi(w)$ je bez čtverců pro každé slovo w bez čtverců s délkou $|w| = 3$.*

Pro reprezentaci tří písmen využíváme v počítači dva bity, čímž ale nevyužijeme těchto dvou bitů naplno. Ditherační posloupnost nad čtyřpísmennou abecedou se proto jeví výhodnější.

Definujme morfismus $\varphi : \{0,1,2,3,4\}^* \to \{0,1,2,3\}^*$ takto

$$\varphi(0) = 0102\,, \quad \varphi(1) \quad = 0123\,, \quad \varphi(2) = 0312\,, \quad \varphi(3) \quad = 1013\,, \quad \varphi(4) = 1032\,.$$

Platí, že $\varphi(w)$ je bez čtverců pro všechny faktory $w \in \mathcal{L}_3(\mathbf{d})$, které neobsahují čtverce. Odtud i $\widetilde{\mathbf{d}} = \varphi(\mathbf{d})$ je bez čtverců s komplexitou $\mathcal{C}_{\widetilde{\mathbf{d}}}(n) \geq 2^{\frac{n}{8}}$.

Ačkoliv se zdá požadavek na malou abecedu ditheračního slova přirozený, nejsnadnější pro praktickou implementaci ditherovaných hašovacích funkcí je využití klasické kompresní funkce F, což nám poskytne pro ditherační písmeno celý jeden byte zkrácením délky vstupního bloku zprávy:

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

V tomto případě nevadí, má-li ditherační slovo abecedu o velikosti až 256 písmen.

Příklad 3. *Jelikož nám nejde o minimální abecedu, můžeme pro zlepšení komplexity zvolit následující postup. Místo prokládání slov* $\mathbf{u}$ *a* $\mathbf{v}$ *můžeme každé dvojici $(u_i, v_i), u_i \in \{0,1\}, v_i \in \{2,3,4\}$ přiřadit nové písmeno $\bar{d}_i$ z abecedy $\{a,b,c,d,e,f\}$ a získat tak slovo $\bar{\mathbf{d}}$ nad šestipísmennou abecedou s komplexitou $\mathcal{C}_{\bar{\mathbf{d}}}(n) \geq 2^n$.*

Při konstrukci vhodných ditheračních slov se vynořují otázky, kterými se klasická kombinatorika na slovech příliš nezabývá:

- Jak sestrojit snadno generovatelné slovo bez čtverců s exponenciální komplexitou nad velkou abecedou (až 256 písmen)?
- Jak snadno generovat slova bez čtverců s dobrým koeficientem v exponentu komplexity nad malou abecedou?
- Jak generovat slova s exponenciální komplexitou rychleji než řazením binárních rozvojů přirozených čísel po sobě?

6 Nový standard SHA-3

Hašovací funkce stojí často na nedokázaných principech, proto je přirozené, že tu a tam dojde k jejich prolomení. Roku 2004 byl prolomen hašovací standard MD5,

v roce 2006 pak známý český kryptolog Vlastimil Klíma publikoval algoritmus hledající kolize MD5 během 1 minuty. I u pozdějšího standardu SHA-1 se kolize umějí hledat rychleji, než je považováno za bezpečné. Proto americký úřad pro standardizaci (NIST) [14] v současné době považuje za bezpečnou hašovací funkci pro kryptografické účely standard SHA-2. Ten je však třikrát pomalejší než SHA-1, a tak NIST v listopadu roku 2007 vyhlásil soutěž o hašovací funkci, která by byla dostatečně rychlá a spolehlivá, aby mohla SHA-2 v budoucnu nahradit. Také byl požadavkem algoritmus naprosto odlišný od SHA-2, aby nevedlo případné prolomení SHA-2 k prolomení SHA-3 či naopak.

Této soutěže se dohromady zúčastnilo 64 hašovacích funkcí od 191 autorů. Mezi nimi byly i dvě funkce s českými spoluautory – EDON-R a BMW (Blue Midnight Wish). Na obou dvou programech pracovali (kromě jiných) Makedonec Danilo Gligoroski a již zmíněný známý český kryptolog Vlastimil Klíma. Oba dva programy patřily mezi nejrychlejší, ukázalo se však, že funkce EDON-R není dostatečně odolná vůči hledání vzoru, a tak do dalšího kola postoupilo spolu s dalšími třinácti kandidáty pouze BMW. V prosinci 2010 vyhlásil NIST pět finalistů, mezi které se již algoritmus BMW nedostal, přestože byl jedním z nejrychlejších a nebyly u něj nalezeny žádné slabiny [9]. Mezi pět postupujících funkcí patřily algoritmy BLAKE, Gröstl, JH, Keccak a Skein. V říjnu 2012 byl vyhlášen vítěz soutěže – algoritmus Keccak. NIST vybral KECCAK, jak oficiálně praví [14], z důvodu jeho elegantního návrhu, velké bezpečnostní rezervy, přizpůsobivosti, dobrého výkonu obecně a výborného výkonu v hardwaru. Citujme ovšem slova V. Klímy [10]: „Připomeňme, že NIST se „odklonil“ od vyhlášených platných požadavků soutěže a uprostřed soutěže je změnil, což také veřejně konstatoval. Ustoupil z požadavku, že nový standard musí být podstatně rychlejší než SHA-2. To NISTu jako jeden ze soutěžících (a spoluautor nejrychlejšího kandidáta v druhém kole) nikdy neodpustím… Nicméně důležité je, že NIST nové SHA-3 věří. Také obavy o bezpečnost SHA-2, panující před soutěží, se nenaplnily. Dokonce se ještě nepodařilo prakticky prolomit SHA-1! To je dobrá zpráva pro nás všechny, neboť průmysl IT se bez kvalitní kryptografie neobejde. Přínosem soutěže bezesporu je, že dnes může průmysl IT na poli hašovacích funkcí být v klidu, neboť máme ve skutečnosti dva standardy SHA-2 a SHA-3 a není pravděpodobné, že by se někomu podařilo prolomit jak SHA-2, tak SHA-3. Vývojáři si dnes mohou vybrat ten algoritmus z rodin SHA-2 a SHA-3, který bude pro ně rychlejší, bezpečnější, méně náročný na paměť, výkon apod. Nemusí přitom pospíchat, protože SHA-2 by mohla být v platnosti ještě cca 10 let a možná i déle.“

Z našeho pohledu ještě konstatujme, že většina soutěžních algoritmů používá iterativní konstrukci založenou na kompresní funkci a brání se útoku proti druhému vzoru hrubou silou. Používá se tzv. dvojnásobná (i vícenásobná) pumpa (anglicky *wide-pipe*), kdy se délka průběžných haší zdvojnásobí, čímž se výpočet hašovací funkce zpomalí zhruba čtyřikrát. Ač je tedy metoda ditherování elegantnější, hrubá síla je v tomto případě pro praktické účely postačující. Nutno ale přiznat, že dvojnásobná pumpa na rozdíl od ditherování zabraňuje kromě útoku na druhý vzor i Jouxovu útoku na multikolize [7].

Reference

[1] Andreeva E., Bouillaguet Ch., Fouque P.-A., Hoch J. J., Kelsey J., Shamir A., Zimmer S., *Second preimage attacks on dithered hash functions*, Advances in Cryptology – EUROCRYPT 2008, Lecture Notes in Computer Science **4965** (2008), 270–288.

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

[3] Biham E., Dunkelman O., *A framework for iterative hash functions - HAIFA*. IACR Cryptology ePrint Archive 2007: **278** (2007)

[4] Brandenburg F.-J., *Uniformly growing k-th power-free homomorphisms*. Theoretical Computer Science **23** (1983), 69–82.

[5] Damgard I., *A design principle for hash functions*, Advances in Cryptology - CRYPTO 1989, Lecture Notes in Computer Science **435** (1989), 416–427.

[6] Dean R. D., *Formal aspects of mobile code security*. PhD thesis, Princeton University, 1999.

[7] Joux A., *Multicollisions in iterated hash functions, application to cascaded constructions*, Advances in Cryptology - CRYPTO 2004, Lecture Notes in Computer Science **3152** (2004), 306–316.

[8] Kelsey J., Schneier B., *Second preimages on n-bit hash functions for much less than 2^n work*, Advances in Cryptology – EUROCRYPT 2005, Lecture Notes in Computer Science **3494** (2005), 474—490.

[9] Klíma V., *Jak dopadla soutěž SHA-3?*, Crypto-World **10** (2010), 2–10.

[10] Klíma V., *Nový standard digitálního otisku SHA-3*, Crypto-World **12** (2012), 5–7.

[11] Menezes A. J., van Oorschot P. C., Vanstone S. A., *Handbook of applied cryptography*. CRC Press, 1996.

[12] Merkle R. C., *A certified digital signature*. In Advances in Cryptology - CRYPTO 1989, Lecture Notes in Computer Science **435** (1989), 218–238.

[13] Rivest R. L., *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)

[14] Domácí stránka soutěže. http://csrc.nist.gov/groups/ST/hash/sha-3

Ing. L'ubomíra Balková, Ph.D.
Katedra matematiky FJFI ČVUT v Praze
Trojanova 13
Praha 2 120 00
lubomira.balkova@fjfi.cvut.cz