Filozofická Fakulta Univerzity Karlovy
Ústav Informačních Studií a Knihovnictví

Vývoj a využití hašovacích funkcí při zpracování informací

Development and utilization of hash functions for information processing

DIPLOMOVÁ PRÁCE

Jana Zimmermannová, BBus.

Praha, 2012

Vedoucí práce: Doc., RNDr. Jiří Ivánek, CSc.

Prohlašuji, že jsem diplomovou práci vypracovala samostatně, že jsem řádně citovala všechny použité prameny a literaturu a že práce nebyla využita v rámci jiného vysokoškolského studia či k získání jiného nebo stejného titulu.

V Praze dne 23. 4. 2012 …….

Jana Zimmermannová

Abstrakt

Koncem 70. let začal vznikat pojem, který dnes označujeme jako kryptografická hašovací funkce. V současnosti jsou tyto funkce spojovány zejména s digitálním podpisem. V roce 2005 byla prolomena celosvětově nejpoužívanější funkce SHA-1. Tato skutečnost vedla k tomu, že v roce 2007 vyhlásil NIST soutěž o vytvoření nového bezpečného hašovacího algoritmu. Práce pojednává problematiku kryptografických hašovacích funkcí od počátku jejich teoretické formulace až po současné dění v této oblasti.

Klíčová slova: Kryptografické hašovací funkce, SHA-1, MD5, soutěž NIST

Abstract

At the end of 70th of last century the concept began to emerge, now is referred as a cryptographic hash function. Currently, these functions are associated especially with a digital signature. In 2005, the worldwide most used function SHA-1 was broken. This fact led in 2007 NIST announced a public competition to create a new secure hash algorithm. This Thesis deals with issues of cryptographic hash functions from the beginning of their theoretical formulation to current events in this area.

Key words: Cryptographic hash functions, SHA-1, MD5, NIST competition

Obsah

1. Úvod ... 8
2. Kryptografická hašovací funkce, její vlastnosti a použití ... 10
   2.1. Hašovací funkce ... 10
   2.2. Historie a souvislosti pojmu ... 12
   2.3. Vztahy mezi vlastnostmi ... 16
   2.4. Útoky, kolize a prolomení ... 19
   2.5. Použití ... 21
   2.6. Shrnutí ... 25
3. Konstrukce hašovací funkce ... 27
   3.1. Merkle-Damgårdova konstrukce hašovací funkce ... 28
   3.2. Další konstrukce hašovací funkce ... 29
   3.3. Konstrukce kompresní funkce ... 34
   3.4. Shrnutí ... 36
4. Současné hašovací funkce ... 38
   4.1. Stručný přehled ... 38
   4.2. Rodina funkcí MDx ... 43
   4.3. Rodina funkcí SHA ... 47
   4.4. Shrnutí ... 53
5. Nový hašovací standard ... 54
   5.1. První a druhé kolo soutěže ... 56
   5.2. Finále soutěže ... 62
   5.3. Shrnutí ... 64
6. Závěr ... 65

Použité zdroje ... 66

Seznam vyobrazení ... 78

Seznam symbolů a zkratek

$\neg$	unární logická spojka negace - NOT
$\wedge$	binární logická spojka konjunkce - AND
$\vee$	binární logická spojka disjunkce - OR
$\oplus$	binární logická spojka exkluzivní disjunkce - XOR
$\neq$	symbol pro nerovnost
$\|\|$	zřetězení argumentů
$\in$	symbol pro náležení prvků do množiny – „je prvkem“
$\subseteq$	množinová inkluze
$\leq$	symbol ostré nerovnosti – „je menší nebo rovno než“
$\geq$	symbol ostré nerovnosti – „je větší nebo rovno než“
$\leftarrow$	zobrazení množiny vpravo do množiny vlevo od symbolu
$\rightarrow$	zobrazení množiny vlevo do množiny vpravo od symbolu
$\| x \|$	absolutní hodnota výrazu x
{ }	slouží pro výčet prvků množiny
$f(x, y)$	obecný zápis funkce, která má argumenty x a y výraz v závorce představuje argument funkce f, jednotlivé argumenty jsou odděleny čárkou
>>>	rotační posun o x bitů vpravo
<<<	rotační posun o x bitů vlevo
>>	posun o x bitů vpravo
mod	značí „modulo“ - zbytek

AES	Advanced Encryption Standard
CRHF	Collision Resistant Hash Function
DES	Data Encryption Standard
FIPS	Federal Information Processing Standard
FRN	Federal Register Notice
GUID	Global Unique Identifier
HMAC	Hash Message Authentication Code
KDF	Key Derivation Function
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code

MDC	Manipulation Detection Code
NIST	National Institute of Standards and Technology
OWHF	One-Way Hash Function
PKCS	Public-Key Cryptography Standard
PRF	Pseudo Random Function
SSH	Secure Shell
SSL	Secure Sockets Layer
TSL	Transport Layer Security
UOWHF	Universal One-Way Hash Function
URN	Uniform Resource Name
UUID	Universal Unique Identifier

1. Úvod

Od doby, kdy Shannon představil svoji informační teorii, uběhlo již mnoho let. V dnešní době již považujeme elektronické zpracovávání informací za něco naprosto běžného a informace zprostředkované touto formou tvoří součást našich každodenních životů.

S nárůstem možností využití počítačového zpracování informací rostla i potřeba nových kryptologických metod, jak tyto informace chránit. Tato potřeba dala za vznik novému kryplologickému primitivu nazývanému „kryptografická hašovací funkce“. Kryptografická hašovací funkce jako element, jenž dokáže z libovolně dlouhého bitového řetězce vytvořit „jedinečný“ řetězec o pevné konstantní délce, v sobě nese prvky jak kryptografie symetrické, tak kryptografie asymetrické a stojí tak na pomezí mezi oběma základními odvětvími kryptologie jako vědní disciplíny.

Ačkoli se může takto deklarovaná skutečnost jevit jako poněkud odtržená od běžného života, opak je pravdou.

Kryptografické hašovací funkce, hovoříme-li o elektronickém zpracování informací, díky svým vlastnostem můžeme nalézt téměř kdekoli. Primárním a zakládajícím prvkem využití těchto funkcí byl digitální podpis, nicméně využití těchto funkcí je mnohem širší a neomezuje se pouze kryptografické účely. Kryptografické hašovací funkce jsou využívány v rámci operačních systémů, kde zajišťují integritu dat, v nejrůznějších síťových protokolech a aplikacích chráněných heslem. Jsou implementovány v knihovnách programovacích jazyků. Používají se pro tvorbu jedinečných identifikátorů.

Jen na základě tohoto drobného výčtu lze odvodit, jak široké spektrum využití těchto funkcí ve skutečnosti je. Možná právě díky tomu jsou tyto funkce někdy trochu nedoceněné, protože představují „pouze“ drobný stavební kámen daleko větších systémů.

Pracujeme s nimi nepřímo téměř všichni, ale málokdo o nich něco ví, případně tuší, že vůbec existují.

Cílem této práce je představit hašovací funkce a jejich vývoj, a to jak z pohledu praktického, tak teoretického. Motivací této práce není analyzovat či nějak hodnotit jednotlivé funkce, nýbrž podat informaci o tom, že existují a jsou nebo byly v praxi užívané.

Minulý čas je dán útoky proti těmto funkcím. Útoky samotné, jedná-li se o ty v rovině řekněme akademické, jsou právě jedním z faktorů, co posouvá vývoj v této oblasti kupředu. Tato práce neposkytuje prostor pro rozebírání jednotlivých útoků, kterých je bezpochyby více než hašovacích funkcí samotných, a není to ani jejím záměrem. Útoky jsou uváděny pouze

jako doklad toho, že určitý hypotetický předpoklad daných funkcí byl vyvrácen a bylo tudíž nezbytné konstruovat funkce nové, teoreticky lepší a odolnější.

Druhá kapitola této práce představuje teoretický vývoj tohoto pojmu jako celku a zároveň na obecné úrovni uvádí možnosti pro jeho využití.

Třetí kapitola představuje některé teoretické konstrukce hašovacích funkcí, což tvoří jakýsi mezistupeň mezi teorií a praxí. Konstrukce uváděné v této kapitole tvoří teoretický podklad pro funkce uváděné v kapitole čtvrté a páté.

Kapitola čtvrtá představuje některé konkrétní, prakticky zkonstruované hašovací funkce v období mezi lety 1989 a 2008. Zvláštní pozornost je věnována dvěma rodinám, které se staly standardy bezpečných hašovacích funkcí, přičemž jednotlivé funkce těchto rodin, jsou celosvětově nejrozšířenějšími a nejpoužívanějšími.

Poslední kapitola je zaměřena na veřejnou soutěž NIST, která v posledních čtyřech letech určovala dění na tomto poli. Díky této soutěži přišlo na svět mnoho nových hašovacích funkcí, byly publikovány některé nové konstrukce, zvýšila se úroveň kryptoanalýzy hašovacích funkcí a celkově tak byl učiněn velký krok kupředu v tomto oboru.

2. Kryptografická hašovací funkce, její vlastnosti a použití

Hašovací funkce v obecném pojetí se začínají objevovat na poli výpočetní techniky někdy v polovině padesátých let minulého století. Jejich využití v oblasti kryptologie však datujeme daleko později. Potřeba vedoucí ke vzniku kryptografických hašovacích funkcí a jejich první teoretická formulace spadá do konce let sedmdesátých.

Cílem této kapitoly je představit termín kryptografická hašovací funkce, uvést vědecké tendence, které daly za znik tomuto pojmu a zároveň vysvětlit vztahy, které v souvislosti s tímto pojmem vyvstávají.

2.1. Hašovací funkce

Obecně je hašovací funkce *h* taková, která zobrazuje bitové řetězce libovolné konečné délky *x* do řetězců pevné délky *h(x)* o velikosti *n* bitů [129]. Zjednodušeně řečeno, je to funkce, která vytváří k libovolně dlouhé zprávě digitální otisk (haš) s pevně definovanou délkou, který je zpravidla několikanásobně menší než původní zpráva.

Zprávou se rozumí jakýkoli odraz skutečnosti v digitální podobě, tedy cokoli od prostého textu až po software, jelikož tyto funkce pracují na binární úrovni. Jedná se tedy o prosté zobrazení *N* binárního prostoru do menšího *n* binárního prostoru. Proto je důležité neomezovat chápání pojmu zpráva pouze na základě jeho sémantického významu.

V současné literatuře je termín hašovací funkce často běžně používán pro kryptografickou hašovací funkci. Jak jsem již v úvodu této kapitoly naznačila, jsou různé techniky hašování, potažmo různé druhy hašovacích funkcí. Před vstupem na pole kryptografie, byly tyto funkce využívány zejména ve vyhledávacích systémech.

Potřeby kryptografie přinesly možnost i jiného využití těchto funkcí, přičemž byly stanoveny určité vlastnosti, které by tyto funkce měli splňovat.

Kryptografická hašovací funkce je tedy taková, která je determinována určitými vlastnostmi.

Nechť jsou definovány tyto vlastnosti:

a) *f zobrazuje vstup x libovolné konečné bitové délky do výstupu $f(x)$ o pevné bitové délce n*
   => **komprese** (compression)
b) *mám-li dáno f a vstup x, $f(x)$ je snadné vypočítat*
   => **jednoduchost výpočtu** (easy computation)
c) *mám-li dáno f a $f(x)$, je výpočetně nemožné nalézt hodnotu x*

=> **jednocestnost/ odolnost vůči nalezení vzoru** (one-way, preimage resistance)

d) je výpočetně nemožné nalézt k x jakékoli $x' \neq x$, *pro které by platilo* $f(x) = f(x')$

=> **odolnost vůči nalezení druhého vzoru/ bezkoliznost 2. řádu** (2nd-preimage resistance, weak collision resistance)

e) je výpočetně nemožné najít libovolnou dvojici různých x a x´, pro kterou platí $f(x)=f(x')$

=> **odolnost vůči kolizím/ bezkoliznost 1. řádu** (collision resistance, strong collision resistance)

Na základě výčtu těchto vlastností mohu říci, že každá funkce mající minimálně vlastnost *a* se dá nazvat jako hašovací a pak kryptografická hašovací funkce je taková, která teoreticky splňuje minimálně první čtyři z výše uvedených vlastností, bez ohledu na použití.

Kryptografická hašovací funkce, jako obecný termín ve smyslu hašovacích funkcí, které jsou využívány v kryptografických systémech, potažmo systémech vyžadujících určitou formu zabezpečení, je tedy jednak „*jednocestná hašovací funkce*“ (slabá jednocestná hašovací funkce, slabě kolizní hašovací funkce), tj. mající vlastnosti *a-d*, „*bezkolizní hašovací funkce*“ (silná jednocestná hašovací funkce), tj. mající vlastnosti *a-e* a takzvaná „*univerzální jednocestná hašovací funkce*“, která může být chápána jako určitý mezistupeň mezi dvěma již uvedenými.

Z hlediska řetězců stojících na vstupu těchto funkcí, jsou kryptografické hašovací funkce rozdělovány na klíčované a neklíčované. Klíčované funkce mají v určitém smyslu blíže k šifrám, jelikož na jejich vstupu stojí dva parametry zpráva a tajný klíč. Výstupy těchto funkcí jsou souhrnně označovány jako MACs (Message Authentication Codes).

Funkce, na jejichž vstupu stojí pouze zpráva, jsou tedy funkcemi neklíčovanými a výstupy těchto funkcí jsou označovány jako MDCs (Manipulation Detection Codes).

V praxi rozdíl mezi funkcemi pro tvorbu MDCs a MACs téměř není. Lépe řečeno zpravidla nejsou konstruovány zvláštní hašovací funkce, které se používají pro detekci manipulace a jiné zvláštní hašovací funkce pro autentikaci zpráv[1].

Ještě lépe řečeno v současné praxi je snahou konstruovat takové funkce, jenž lze využít jak pro tvorbu MAC tak MDC kódů.

Další dělení kryptografických hašovacích funkcí je podle základu, na kterém pracují. Z tohoto pohledu rozlišujeme mezi funkcemi založenými na blokových šifrách, algebraických strukturách a speciálně navrženými (případně uživatelsky navrženými). Speciálně navrženými

[1] Příkladem funkce vytvořené výhradně pro tvorbu MAC kódů je třeba CBC-MAC [116, s. 456].

jsou míněny takové funkce, jejichž základ tvoří kombinace různých metod a nelze tedy říci, že jsou striktně založeny na právě jednom prvku z výše uvedených.

2.2. Historie a souvislosti pojmu

Počátky teoretické formulace kryptografického využití hašovacích funkcí nalezneme v práci Diffieho a Hellmana z roku 1976 [33]. Autoři zde uvádí, že v rámci procesu tvorby systému elektronické formy komunikace je nutné nalézt „digitální fenomén“ se stejnými vlastnostmi jako je ruční podpis, který by zaručil jednoznačnou autentizaci odesilatele. Takovýto podpis musí být pro každého jednoznačně rozpoznatelný, avšak schopnost produkovat jej, může mít pouze jeho původce, tedy odesílatel zprávy.

Řešení vidí v technice, kterou nazývají „jednocestná autentizace“ [33, s. 649]. Uvažují zde funkci f, která by umožňovala pro jakýkoli argument x snadno vypočítat odpovídající hodnotu $f(x) = y$, přičemž při znalosti hodnoty y a znalosti funkce f by bylo výpočetně nemožné získat hodnotu x. Taková funkce je tedy nazývána jednocestná.

Důležitým aspektem, jak uvádějí, je, že noninvertibilitu funkce (tedy nemožnost zpětně získat hodnotu x při znalosti y a f aplikací inverzního postupu) je nutno chápat jinak, než jak je tento pojem běžně užíván v matematice. „Funkce f je normálně nazývána *noninvertibilní* pokud inverze bodu y není jednoznačná, tj. existují různé body x_1 a x_2 pro které platí: $f(x_1) = y = f(x_2)$. Toto není druh inverzní obtížnosti, kterou bychom požadovali. Spíše musí být velice obtížné, mám-li dánu hodnotu y a znalost f, vypočítat naprosto jakékoli x se znalostí $f(x) = y$ [33, s. 650].“

Dalším faktorem, který v této práci nalezneme, je předpoklad náhodného chování funkce f. Náhodné chování funkce je velmi důležité vzhledem k faktu, že je mnoho různých argumentů vedoucích ke stejnému výsledku $f(x)$, jak autoři uvádějí, je jich pravděpodobně 2^{N-n}. Dále v práci říkají, že není-li funkce jednoznačně invertibilní není nutné nalézt konkrétně použité x nýbrž postačí najít jakékoli x vedoucí ke stejnému výsledku [33, s. 650-651].“.

V zásadě v této práci můžeme nalézt všechny podklady pro teoretickou formulaci kryptografické hašovací funkce, resp. formulaci jejich vlastností.

První definici kryptografické hašovací funkce nalezneme v práci Merkla [75, 76]. „Existuje mnoho případů kdy je potřeba autentizovat velkou oblast dat, avšak pouze malá oblast dat může být uložena nebo autentizována. …Tím roste potřeba jednocestné hašovací funkce. Intuitivně, jednocestná hašovací funkce F bude taková, pro kterou platí:

1. *F* může být aplikována na jakýkoli argument o jakékoli délce. Aplikace *F* na více než jeden argument (tj. $F(x_1, x_2)$) je ekvivalentní jako aplikace *F* na zřetězení argumentů, tj. $F(<x_1, x_2>)$.
2. *F* vždy produkuje výstup o fixní délce.
3. Mám-li dáno *F* a *x*, mohu snadno vypočíst hodnotu *F(x)*.
4. Mám-li dáno *F* a *F(x)*, je výpočetně nemožné určit hodnotu x.
5. Mám-li dáno *F* a *x*, je výpočetně nemožné nalézt $x' \neq x$, pro které platí $F(x) = F(x')$.

Hlavním využitím jednocestných funkcí je autentizace. Pokud může být autentizováno *y*, můžeme autentizovat x prostřednictvím výpočtu *F(x)* [75, s. 11–12].“

Tento poznatek je velmi důležitý. Zpráva může být velmi dlouhá, kdybychom podepisovali přímo zprávu, bylo by to operačně velmi náročné, existuje-li však funkce, která je schopna vytvořit „jedinečný“ reprezent této zprávy podstatně menších rozměrů, je možné k podpisu využít pouze tento reprezent.

Zde bych chtěla upozornit, že výstup hašovací funkce, neboli-li otisk, není totéž jako digitální podpis, je to pouze jeden z jeho stavebních prvků. Slovo jedinečný jsem dala záměrně do uvozovek, protože víme, že ve skutečnosti jedinečný není, resp. víme, že teoreticky může existovat velmi mnoho zpráv, které mohou vést ke stejnému otisku.

Základní předpoklad hašovacích funkcí nespočívá v tom, že neexistují dvě (nebo více) různé zprávy, kterým by funkce přiřazovala stejný výstup, předpoklad spočívá v tom, že takovéto zprávy je výpočetně nemožné nalézt.

Jak Merkle [75, s. 12] ve své práci pozoroval, je-li dána pevná délku výstupu *n*, je limitován počet možných výstupů touto délkou, tedy počet všech výstupů je roven 2^n. Potom je-li znám určitý počet různých výstupů *k*, postačí k nalezení zprávy vedoucí ke stejnému otisku 2^{n-k} pokusů. Z toho vyplývá, že samotná délka výstupu hraje významnou roli ve vztahu k bezpečnosti funkce.

Další významný vztah vzhledem k bezpečnosti je, že složitost nalezení dvou různých vstupů vedoucích ke stejnému otisku by měla být větší než $2^{n/2}$ a složitost nalezení druhé zprávy vedoucí na stejný otisk jako zpráva originální by měla být 2^{n-1} [76, s. 220]. Vztah 2^{n-1} je v celku snadno pochopitelný. Říká, že pro nelezní druhé zprávy, která bude mít stejný otisk jako zpráva původní, bude v ideálním případě potřeba takový počet pokusů, který je roven počtu všech možných výstupů funkce.

Zajímavé však je, že Merkle zde dospěl nezávisle ke stejnému závěru jako Yuval, a sice, že k nalezení dvou zpráv vedoucí ke stejnému otisku bude s vysokou pravděpodobností stačit

takový počet pokusů, který je roven druhé odmocnině z celkového počtu možných výstupů funkce, tedy $2^{n/2}$. Proto je v současnosti považována polovina délky výstupu za hranici bezpečnosti hašovací funkce.

Yuval [132] v roce 1979 publikoval útok, který se stal pravděpodobně nejslavnějším útokem v historii kryptografických hašovacích funkcí. Do historie se zapsal pod názvem „*Square-root Attack*“ (útok odmocninou) nebo ještě známějším názvem „*Birthday Attack*“ (narozeninový útok). Onen útok spočíval v generování různých zpráv se stejným sémantickým významem. Princip útoku je zhruba následující: Útočník má falešnou zprávu *x´*, kterou si přeje, aby oběť útoku podepsala, a pravou zprávu *x*, o které ví, že se jí oběť chystá podepsat (případně je ochotná tak učinit). Poté generuje různé verze zprávy *x* a *x´*, tak aby byl zachován jejich sémantický význam, do doby dokud nenarazí na dvě takové verze x_n a $x_n´$, které by měly stejný otisk. Oběti je pak předložena k podpisu zpráva x_n a takto získaný podpis je pak přenesen na falešnou zprávu $x_n´$, která může být vydávána za pravou.

Yuval přišel s tímto útokem, jako reakcí na koncept digitálního podpisu, který publikoval Rabin [100, 101], a ve kterém jako první používá hašovací funkci.

V Merkleho definici vidíme, že zde ještě není zohledněn vztah mezi zprávami x a x´, kdy x a x´ jsou dvě jakékoli libovolné zprávy. Merkle tedy definoval to, co nazýváme „jednocestná hašovací funkce“. Jak ukázal Yuvalův útok, nejen že by nemělo být „možné“ nalézt druhou zprávu vedoucí ke stejnému otisku, ale nemělo by být „možné“ nalézt jakékoli dvě zprávy s tímtéž otiskem, protože zde skutečně platí, že $h(x) \neq h(x_n´)$ a zároveň $x \neq x_n \neq x_n´$.

Jako první vnáší formální požadavek na bezkoliznost do definice Damgård [32]. V roce 1987 přichází s konceptem bezkolizních hašovací funkcí, kde zvažuje rodinu bezkolizních funkcí s parametrem *k*, který by udával hodnotu bezpečnosti.

„Rodina bezkolizních hašovacích funkcí je množina hašovacích funkcí s následujícími vlastnostmi:

Existuje pravděpodobnostní polynomiální časový algoritmus[2], který na vstupu, na základě hodnoty bezpečnosti vybere jednotně a náhodně člena rodiny s touto hodnotou.

Všechny funkce v rámci rodiny jsou výpočetně řešitelné v polynomiálním čase.

Problém nalezení $x \neq y$ stejně jako $h(x) = h(y)$, pro dané *h* z rodiny, je výpočetně nemožný. Člen bezkolizní rodiny je tedy nazýván bezkolizní.

Pokud je *h* bezkolizní, potom funkce *h* je také jednocestná v následujícím smyslu:

[2] Polynomiální čas – Čas definovaný polynomiální složitostí. Polynomiální složitost je třídou asymptotické složitosti, která představuje je způsob klasifikace počítačových algoritmů.

Je dána bezkolizní rodina a člen rodiny $h: M \rightarrow A$. Pro každé konečné $W \subseteq M$ s absolutní hodnotou $|W| - |A|$ rovnou ne nezanedbatelnému zlomku $|A|$, je výpočetně nemožné pro dané $h(w)$, kde $w \in W$, najít jakékoli $w' \in W$ s $h(w') = h(w)$, pro více než nezanedbatelný zlomek w z množiny W [32, s. 205] .“ Jen pro upřesnění M je množina všech konečných slov, W je konečná množina všech možných zpráv a A je konečná množina všech možných zobrazení.

Damgårdova definice je celkem zdlouhavá. Méně formální, zato daleko jasnější, definici lze nalézt u Merkla [77]. Ten říká že, ve „slabých“ jednocestných hašovacích funkcích není garance, že bude výpočetně nemožné najít dvojici z a z', které povedou ke stejnému zobrazení, ačkoli bude platit $x \neq z$ a $x \neq z'$. Musí být dokonce obtížné generovat mnoho dvojic z a z', jinak ani náhodně vybrané x nemůže být v bezpečí.

Proto definuje „silnou“ jednocestnou hašovací funkci takto:

1. Funkce F může být aplikována na jakýkoli argument o jakékoli délce. F aplikovaná na více než jeden argument, je rovna aplikaci F na bitové zřetězení argumentů.
2. F produkuje výstup o pevné délce.
3. Mám-li dánu funkci F, je snadné vypočítat $F(x)$.
4. Mám-li dánu funkci F, je výpočetně nemožné najít jakýkoli pár x, x' takové, pro které platí $x \neq x'$ a $F(x) = F(x')$.

Abych tuto podkapitolu uzavřela, na počátku devadesátých let minulého stolení, přichází ještě Naor a Yung [83] s konceptem „univerzální jednocestné hašovací funkce“. Jedná se v podstatě o rodinu jednocestných hašovacích funkcí s bezpečnostním parametrem podobně, jak uváděl Damgård [32].

Tyto funkce také poskytují bezkoliznost, ale trochu jiným způsobem. Myšlenka spočívá v tom, že je sice možné nalézt kolizi pro funkci s konkrétním parametrem („klíčem“), ale ne v rámci libovolného užití funkce.

Tato koncepce není ovlivněna tak zvaným „Narozeninovým paradoxem“, jelikož striktně předpokládá, že nejprve musí být vytvořena zpráva a poté je na jejím základě náhodně vybrána konkrétní funkce (resp. klíč) pro zpracování. Případný útočník tedy předem nemůže vědět, jaká funkce bude pro zpracování použita, a tudíž nemůže generovat zprávy, tak jak to popisuje Yuvalův útok. Trochu problém je však v tom, jak v praxi docílit, aby toto pořadí bylo dodrženo.

2.3. Vztahy mezi vlastnostmi

Jelikož zavedení hašovacích funkcí do kryptologie souviselo s vývojem schématu pro digitální podpis, jsou definované vlastnosti těchto funkcí a jejich potřeba, pro dosažení určité bezpečnosti konceptu, celkem snadno pochopitelná. Problémem však je nejednotné chápání definic těchto vlastností, což částečně odráží i nejednotnost v jejich názvech.

„Zatímco termín „one-way“ (jednocestnost) je obecně brán ve významu „preimage resistant“ (odolnost vůči nalezení vzoru), v literatuře je někdy užíván k implikaci toho, že funkce je „2nd-preimage resistant“ (odolná vůči nalezení druhého vzoru) nebo výpočetně noninvertibilní. Což způsobuje určité nejasnosti ve smyslu, že „odolnost vůči nalezení druhého vzoru“ nezaručuje „odolnost vůči nalezení vzoru“ a ani „odolnost vůči nalezení vzoru“ nezaručuje „odolnost vůči nalezení druhého vzoru“ [74, s. 329].“

Formálně se definicí vlastností a vztahů mezi nimi zabývají Rogaway a Shrimpton v [112]. Poukazují zde na fakt, že neformální definování vlastností hašovacích funkcí může vést k řadě nejasností, a tudíž tyto neformální definice lze vykládat různými způsoby, které mohou být považovány za pravdivé a nepravdivé zároveň. Formalizace se týká zejména dvou výše uvedených vlastností, tedy skutečnosti za jakých podmínek lze hašovací funkci považovat za odolnou vůči nalezení vzorů. Dospěli k názoru, že je nutné brát celou situaci z pohledu osoby, která vede útok proti hašovací funkci[3]. Útočník má pro útok možnost náhodné volby funkce na základě určitého parametru „klíče“, náhodné volby zprávy (respektive jejího otisku) nebo náhodné volby obou faktorů.

Čímž rozšířili definice vlastností o stavy, kdy je funkce odolná „vždy“ (fixní klíč, náhodná zpráva) a „kdekoli“ (fixní zpráva, náhodný klíč).

Konkrétně jejich formální definice pro jednocestnost (odolnost vůči nalezení vzoru) vypadá takto:

Nechť $H = K \times M \rightarrow Y$ je rodina hašovacích funkcí a m je číslo takové, pro které platí $\{0,1\}^m \subseteq M$. Nechť A značí “adversary” (útočník). Potom definujeme:

$$Adv_H^{\mathrm{Pre}[m]}(A) = Pr\,[K \leftarrow K;\ M \leftarrow \{0,1\}^m;\ Y \leftarrow H_K(M);\ M' \leftarrow A\,(K,Y):\ H_K(M') = Y]$$

$$Adv_H^{e\,\mathrm{Pre}[m]}(A) = \max_{Y \in \gamma} \{Pr[K \leftarrow K;\ M \leftarrow A(K):\ \mid\ H_K(M) = Y]\}$$

$$Adv_H^{a\,\mathrm{Pre}[m]}(A) = \max_{K \in \kappa} \{Pr[M \leftarrow \{0,1\}^m;\ Y \leftarrow H_K(M);\ M' \leftarrow A\,(Y):\ H_K(M') = Y]\}$$

[3] Útočníkem nemusí být nutně fyzická osoba, tedy alespoň ne přímo. Rjačko [111] definuje útočníka jako stroj s náhodným přístupem (Random Access Machine).

První definice představuje obecně formální zápis jednocestné hašovací funkce. Tedy vlastnost, že při znalosti libovolného náhodně vybraného klíče a libovolného náhodně vybraného otisku útočník nesmí být schopen rekonstruovat *M* prostřednictvím funkce *H*. Druhá definice říká, že je-li funkce odolná vůči nalezení vzoru „kdekoli“ (e značí „everywhere“), tak je výpočetně obtížné z jakéhokoli libovolného *Y* (z množiny všech možných) při znalosti konkrétního (tedy nevybraného náhodně) klíče *K* nalézt hodnotu *M*. Poslední definice říká, že je-li funkce odolná vůči nalezení vzoru „vždy“ (a značí „always“), tak je výpočetně nemožné při znalosti libovolného klíče *K* nalézt pro konkrétní *Y* hodnotu *M*. Oním klíčem je pro neklíčované hašovací funkce, chápána iniciační hodnota (viz. kapitola 3). Jelikož iniciační hodnotu zpravidla tvoří průběžný výsledek hašování, je tedy *K* určitým bodem zpracování funkce.

Obdobně je definována i odolnost vůči nalezení druhého vzoru:

$$Adv_H^{Sec[m]}(A) = Pr\,[K \leftarrow K;\ M \leftarrow \{0,1\}^m;\ M' \leftarrow A(K,M):\ (M \neq M') \wedge (H_K(M) = H_K(M'))]$$

$$Adv_H^{eSec[m]}(A) = \max_{M \in \{0,1\}^m} \{Pr[K \leftarrow K;\ M' \leftarrow A(K):\ (M \neq M') \wedge (H_K(M) = H_K(M'))]\ \}$$

$$Adv_H^{aSec[m]}(A) = \max_{K \in \kappa} \{Pr[M \leftarrow \{0,1\}^m;\ M' \leftarrow A(M):\ (M \neq M') \wedge (H_K(M) = H_K(M'))]\}$$

První definice je opět obecná a předpokládá zcela náhodnou volbu faktorů. Druhá definice (eSec), tedy kdy je funkce odolná vůči nalezení druhého vzoru kdekoli, představuje vlastnost, že je obtížné nalézt jakoukoli druhou zprávu (z množiny všech možných), která by při znalosti libovolného (náhodně vybraného) klíče *K* vedla ke stejnému otisku jako originální zpráva *M*. Tato definice představuje definici univerzální jednocestné funkce, jak ji vytvořil Naor a Yun [83]. Poslední definice opět zesiluje definici předchozí, a říká, že při znalosti jakéhokoli (konkrétně vybraného) *K* a zprávy *M* útočník nesmí být schopen nalézt druhou zprávu (různou od *M*), která by vedla ke stejnému konkrétnímu otisku jako *M*.

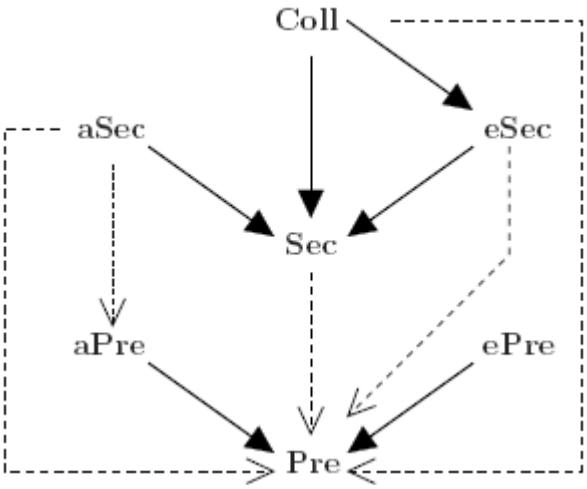
Tedy na základě znalosti průběžných výsledků hašování, zprávy *M* a funkce samotné útočník nesmí nalézt druhý vzor zprávy (tedy nějakou jinou zprávu *M´*), která by měla stejný otisk jako *M*.

Rozšiřovat definici bezkoliznosti by nemělo smysl z prostého důvodu, a sice, že bezkoliznost obecně předpokládá dvě jakékoli různé zprávy *M* (tedy náhodně vybrané), tudíž není možné předem vybrat konkrétní klíč. Ve chvíli, kdy vyberu konkrétní klíč, na jehož základě budu hledat dvě jakékoli zprávy vedoucí ke stejnému otisku, již nehledám kolizi, ale druhý vzor, protože jedna zpráva se stává vzorem druhé.

Pouze pro úplnost tedy formální definice bezkoliznosti zní:

$$Adv_H^{Coll}(\mathrm{A}) = \Pr[K \leftarrow K; (M,M') \leftarrow A(K) : (M \neq M') \wedge (HK(M) = HK(M'))]$$

Na základě těchto výše uvedených definic, odvodili vztahy mezi jednotlivými vlastnostmi, které znázorňuje následující schéma (Obrázek 1).



Obrázek 1 Vztahy mezi vlastnostmi hašovacích funkcí [112, s. 4].

Ze schématu vyplývá, že bezkoliznost vždy implikuje odolnost vůči nalezení druhého vzoru a to kdekoli. Je-li funkce odolná vůči nalezení druhého vzoru kdekoli a vždy, je tedy odolná vůči nalezení druhého vzoru obecně. To samé platí pro nalezení vzoru, resp. je-li funkce odolná vůči nalezení vzoru kdekoli a vždy, je funkce jednocestná.

Vztahy mezi vlastnostmi označené přerušovanou čarou značí, že daná vlastnost může implikovat druhou v závislosti na relativní velikosti výše uvedených množin.

Práci Rogawaye a Schrimptona ještě rozšiřuje Rjačko [111].

Ten přidává do vztahů k základním bezpečnostním vlastnostem hašovacích funkcí ještě „nepadělatelnost“ (unforgeability), vlastnost pseudo-náhodné funkce a pseudo-náhodného orákula. Nepadělatelnost je vztažena ke kódům typu MAC. Dvě poslední vlastnosti by měli přiblížit hašovací funkci k modelu náhodného orákula[4], tedy jakéhosi ideálního stavu náhodného a tudíž nepředvídatelného chování.

To vychází z faktu, že chová-li se funkce náhodně, je nalezení jakékoli kolize, tudíž i vzoru, velmi nepravděpodobné.

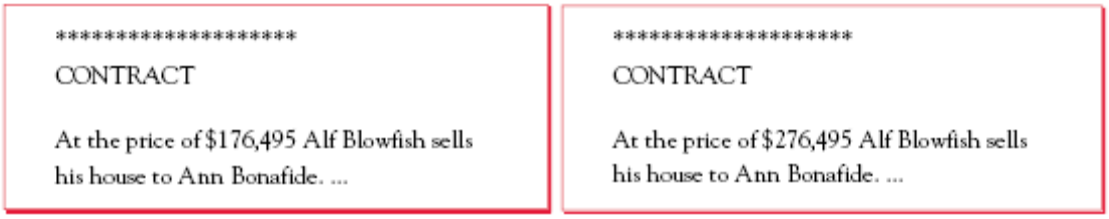
[4] Problematikou modelu náhodného orákula se zabývá např. Bellare a Rogaway [6] nebo Mauer, Renner a Holenstein [71].

2.4. Útoky, kolize a prolomení

Jak z předchozích částí vyplývá, o kolizi hovoříme ve chvíli, kdy jsou nalezeny dvě různé zprávy, jejichž zpracování hašovací funkcí vede ke stejnému otisku. V souvislosti s praktickou konstrukcí a praktickým využitím hašovacích funkcí vstupují do hry ještě další termíny, a sice významná kolize, blízká kolize, pseudo-kolize, blízká pseudo-kolize a multikolize.

Pod termínem multikolize se rozumí nalezení *r* různých zpráv vedoucí ke stejnému otisku.

Z pohledu bezkoliznosti je nepodstatné zda kolidující zprávy (v případě, že zprávou je nějaké skutečné sdělení) mají či nemají sémantický význam. Lépe řečeno zprávy mohou být absolutně nesmyslné. O významné kolizi hovoříme ve chvíli, kdy jsou nalezeny takové zprávy, jejichž sémantický význam je zaměnitelný. Příkladem takovéto významné kolize může být třeba útok, který publikoval Dobbertin [34].



Obrázek 2 Kolize MD4 [34, s. 5]

Dalším pojmem je blízká kolize. Blízká kolize nazýváme stav, kdy otisk dvou zpráv se liší pouze v malém počtu bitů. Ačkoli nalezení blízkých kolizí nepředstavuje prolomení hašovací funkce, zvyšuje pravděpodobnost nalezení kolize skutečné, čímž snižuje její bezpečnost, jelikož ukazuje, že funkce se nechová tak náhodně jak se očekává.

Pojmy pseudo-kolize a blízká pseudo-kolize souvisí s vlastní konstrukcí hašovací funkce (blíže kapitola 3). Jako pseudo-kolize je označována kolize na kompresní funkci, tj. kolize na průběžné hašovací hodnotě, nikoli na výstupu hašovací funkce jako celku. Do kompresní funkce vstupuje zpráva a určitá iniciační hodnota. Pseudo-kolize nastává ve chvíli kdy dvě různé zprávy (resp. části zprávy) s různými iniciačními hodnotami vedou ke stejnému výstupu kompresní funkce. Z toho pak blízké pseudo-kolize, stejně jako blízké kolize, jsou takové, kde se průběžný haš, liší pouze v malém počtu bitů. Pseudo-kolize mohou být velmi nebezpečné, zejména pro funkce využívající Merkle-Damgårdovu konstrukci.

S kolizemi souvisí pojem útok. Útoky na hašovací funkce jsou právě tím, co posouvá v tomto směru vývoj kupředu. Útoky můžeme rozdělovat podle různých hledisek. Ve smyslu vlastností hašovacích funkcí můžeme dělit útoky na „Preimage Attacks“, „Second-preimage

Attacks“ a „Collision Attacks“. Útoky zaměřené na nalézání kolizí jsou, jak již lze logicky odvodit ze samotné definice bezkoliznost, nejčastější, právě proto, že nalézání kolizí je „snadnější“ než nalézání vzoru zprávy.

Podle způsobu vedení útoku můžeme rozlišovat útoky hrubou silou nebo útoky na základě diferenční kryptoanalýzy.

Z hlediska směřování útoku rozlišujeme útoky na cílené a necílené. Cílené útoky jsou takové, které jsou zaměřené buď na určitou komponentu hašovací funkce, nebo na určité použití hašovací funkce. V případě komponenty je to buď celá, nebo určitá část (např. určité kolo zpracování) kompresní funkce. Příkladem cílovaného útoku na konkrétní použití jsou takzvané „Rainbow Tables“, které představují určitý seznam zpráv a otisků pro danou hašovací funkci. Tyto tabulky se používají ke zjišťování hesel do systému, která jsou uložená ve formě otisku.

Obecně cílem každého útoku je prolomení hašovací funkce. Kdy tedy nastává prolomení hašovací funkce a co to přesně znamená?

Hovoříme-li o prolomení hašovací funkce rozlišujeme mezi kryptografickým, teoretickým a praktickým prolomením. Kryptograficky je funkce prolomena ve chvíli, je-li nalezen efektivní způsob nalézání kolizí a vzorů, který je rychlejší než ideální bezpečnost funkce, daná předpokladem, že se funkce chová jako náhodné orákulum. Konkrétně tedy, jak již bylo uvedeno v předchozí podkapitole, ideální bezpečností je složitost větší než $2^{n/2}$ pro nalezení kolize a 2^{n-1} pro nalezení vzorů, kde *n* přestavuje délku výstupu hašovací funkce. V případě, že je nalezena metoda hledání kolizí s menší složitostí než $2^{n/2}$, neznamená to, že funkce nemůže být dále nějakým způsobem používána. Znamená to „pouze“, že funkce není bezkolizní, a tudíž by se neměla používat v takových systémech, které bezkoliznost striktně z hlediska bezpečnosti vyžadují.

Například pro konstrukci autentizačních kódů HMAC, nebo pro funkce odvozující klíče blokových šifer prostřednictvím hašovacích funkcí (KDF), není bezkoliznost hašovací funkce až tolik podstatná. Avšak pro konstrukci digitálních podpisů představuje porušení bezkoliznost značné riziko.

Problémem je, že je-li vyvrácena hypotéza o bezkoliznosti funkce, zvyšuje se i pravděpodobnost nalezení vzorů, což je dáno vztahem mezi vlastnostmi a v konečném důsledku může teoreticky vést až k vyvrácení jednocestnosti funkce.

Praktické prolomení hašovací funkce nastává ve chvíli, kdy naleznu dvě zprávy vedoucí na stejný otisk. Čistě teoreticky mohu zkoušet vytvářet různé zprávy a generovat jejich otisky a

po určité době, resp. po určitém počtu pokusů, zákonitě musím přijít minimálně na dvě, které budou mít stejný otisk. Takovýto útok je nazýván útok hrubou silou. Prakticky tedy naleznu kolizi, ale bezpečnost funkce to nijak neohrozí. Praktické prolomení hašovací funkce tedy nelze chápat jako jakékoli nalezení kolize, či vzoru, nýbrž takové, které je nějakým způsobem systematicky podložené.

Teoretické prolomení nastává ve chvíli, kdy je nalezen takový algoritmus, který umožňuje nalézt kolizi v rozumném časovém horizontu. Přičemž takovýto útok musí být prakticky realizovatelný.

Teoreticky mohu přijít s takovým algoritmem, který umožňuje velmi rychlé nalézání kolizí, avšak útok není prakticky konstruovatelný, jelikož dostupná úroveň výpočetní techniky to z nějakého důvodu neumožňuje. Čili kryptograficky je funkce prolomena, avšak teoreticky ani prakticky nikoli.

Velmi názorně popisuje vztah mezi kryptografickým, teoretickým a praktickým prolomením hašovací funkce Vondruška [124] na funkci SHA-1. „Funkce byla kryptograficky prolomena, neboť byl nalezen postup hledání kolizí se složitostí nižší než 2^{80}. … Množství operací 2^{69} je však stále ještě vysoké, a tak mnoho lidí a institucí nadále obhajovalo SHA-1 jako použitelnou a to pro velkou teoretickou a praktickou náročnost prolomení. …Krátce na to v září 2005 byla SHA-1 teoreticky prolomena. Akashi Satoh publikoval práci, která obsahuje návrh hardware, který by nalezl kolize podle postupu Wangové se složitostí 2^{69}. Navrhuje se architektura LSI na bázi 0.13-μm CMOS. Na základě toho byla vypočítána rychlost, velikost a spotřeba HW. Za 10 milionů dolarů lze tak teoreticky sestavit zákaznický hardwarový systém, který by sestával z 303 PC, každý s 16 deskami (na každé je 32 jader SHA-1), pracujícími paralelně. Útok by trval 127 dní. Takže již zbývá poslední krok – zveřejnění, informace, že SHA-1 byla prakticky prolomena [124, s. 4]!„

Z uvedeného textu vyplývá, že kryptografické prolomení vede k teoretickému prolomení a to následně k praktickému prolomení. Z hlediska bezpečnosti hašovacích funkcí je tedy závažnější fakt, že je známa metoda jak nalézat kolize pro danou funkci, než skutečnost zda je tato metoda v současné chvíli prakticky realizovatelná.

2.5. Použití

Hašovací funkce jsou využívány v mnoha oblastech. Primárním účelem hašovacích funkcí je bezpečnost, a to jak směrem navenek od systému, tak uvnitř systému.

Prakticky je možno říci, že každý kdo pracuje s počítačem, ve smyslu běžného uživatele, pracuje nepřímo i s hašovacími funkcemi. V operačních systémech jsou hašovací funkce (mimo jiné) používány pro kontrolu integrity a kontrolu modifikace souborů. V systémech, které vyžadují přihlašování, jsou funkce používány ke kontrole a ukládání hesel.

Hašovací funkce jsou implementovány v mnoha kryptografických standardech a protokolech umožňujících bezpečnou komunikaci na síti.

NIST na svých webových stránkách zveřejňuje seznam legalizovaných kryptografických modulů od roku 1995, které využívají standardizované hašovací funkce. Tento seznam čítá zhruba 1700 výrobců hardwaru a softwaru a tisíce validovaných kryptografických modulů.

Využití funkcí je opravdu mnoho a konkrétních použití ještě více. Shrnuto alespoň do základních bodů:

⇒ **Digitální podpisy a certifikáty**

Digitální podpisy a certifikáty představují hlavní využití hašovacích funkcí a jsou upraveny různými standardy. Příkladem je třeba americký standard pro digitální podpis FIPS PUB 186 [42] nebo obecně mezinárodní norma ISO/IEC 9796[5].

Proces digitálního podepisování je založen na hašovací funkci a asymetrickém šifrování. Zjednodušeně: Původce vytvoří zprávu, ze které pomocí hašovací funkce vygeneruje její otisk. Otisk je zašifrován pomocí veřejného klíče původce. Veřejný klíč je obecně známý a jeho prostřednictvím lze identifikovat původce zprávy. Příjemce zprávy dešifruje otisk pomocí tajného klíče. Je vytvořen nový otisk zprávy, který je porovnán s otiskem získaným po dešifrování. Jsou-li oba otisky shodné, zprávu lze považovat za původní.

Zde je velmi důležitá bezkoliznost hašovací funkce. V případě existence dvou zpráv se shodným otiskem by nebylo možné prokázat pravost zprávy, jelikož klíč, kterým je zpráva šifrována, je veřejný a tudíž dostupný každému.

[5] ISO/IEC 9796 je obecně norma pro schéma digitálního podpisu s obnovou zprávy. Je tvořena několika částmi, které byly v průběhu let různě novelizovány a upravovány, přičemž jednotlivé části upravují různý mechanismus tvorby digitálního podpisu. V rámci této normy jsou různě implementovány hašovací funkce podle ISO/IEC 10118.

ISO/IEC 10118 obecně upravuje hašovací funkce. ISO/IEC 10118-1 specifikuje obecný model hašovací funkce, ISO/IEC 10118-2 specifikuje hašovací funkce založené na n-bitových blokových šifrách, přičemž tyto šifry jsou specifikovány v ISO/IEC 18033-3. ISO/IEC 10118-3 definuje speciálně navržené hašovací funkce, což v současně platné verzi z roku 2004 [56] jsou: RIPEMD-160, RIPEMD-128, SHA-1, SHA-256, SHA-512, SHA-384 a WHIRLPOOL. ISO/IEC 10118-4 specifikuje hašovací funkce založené na modulární aritmetice. Opět, každá z těchto částí je upravována a novelizována zvlášť.

Z hlediska certifikátů je to například standard RFC 3280 [106], který upravuje infrastrukturu certifikátů X.509 založených na šifrování veřejným klíčem pro potřeby sítě internet. Tyto certifikáty jsou vyžadovány např. protokoly IPSec, SSH nebo LDAP.

⇒ **Protokoly pro bezpečnou komunikaci**

Z protokolů je to například SSL/TSL, SSH nebo IPsec umožňující bezpečnou komunikaci na internetu. Konkrétně např. IPsec pro Windows Server 2003 podporuje HMAC-MD5-96 a HMAC-SHA-1-96 [78].

V zásadě lze říci, že v každém zabezpečeném protokolu pro komunikaci můžeme nalézt nějakým způsobem implementované hašovací funkce.

⇒ **Autentizační kódy**

Hašovací funkce se používají pro tvorbu tzv. HMAC kódů. Tyto kódy slouží jednak k zabezpečení zprávy během přenosu a jednak k autentizaci zprávy. Jedná se o kódy, které jsou tvořené hašovací funkcí, která má na vstupu zprávu M a tajný klíč K. V praxi jsou HMAC označovány jako HMAC-„název hašovací funkce“ (např. HMAC-SHA-1). Výpočet HMAC kódů vypadá následovně:

$$HMAC\text{-}H(M, K) = H((K \oplus opad) \,||\, H((K \oplus ipad) || M))$$

Zprávou je zde zpravidla míněna výzva pro server (challenge), jelikož tyto kódy jsou používány zejména v zabezpečených protokolech pro komunikaci.

Definování HMAC kódů můžeme nalézt například v standardech FIPS PUB 198 [43] a RFC 2104 [105].

⇒ **Pseudo-náhodné generátory**

Hašovací funkce jako pseudonáhodný generátor slouží k vytváření dlouhých „náhodných“ bitových řetězců. Pseudo-náhodná posloupnost je získávána zřetězením výstupů hašovací funkce. Pseudo-náhodné generátory jsou používány v případech, kdy máme k dispozici „krátkou“ bitovou posloupnost, z níž potřebujeme získat pseudo-náhodnou posloupnost o velké délce.

⇒ **Derivace klíčů**

V rámci standardu PKCS #5 [92] jsou hašovací funkce používané k odvození klíče z hesla pro systémy založené na symetrické kryptografii, nebo pro funkce vytvářející kódy typu MAC. V rámci tohoto standardu je používaná funkce MD2, MD5 nebo SHA-1. Princip odvození klíče spočívá v několikanásobné aplikaci (min. 1000x) hašovací funkce na heslo, ke kterému je

přidána „sůl“ (pseudonáhodný bitový řetězec), čímž získám pozitivní číslo o počtu 16 (pro MD2 a MD5) nebo 20 (pro SHA-1) znaků (tedy 64 nebo 80 bitů), které tvoří klíč.
Algoritmus derivace klíče za pomoci hašovací funkce je definován takto:

$$T_1 = Hash(P \| S) => T_2 = Hash(T_1) .... T_c = Hash(T_{c-1}), DK = T_c <0..dkLen-1>$$

Výstup celého procesu je na konci zkracován na požadovanou délku stanovenou pro danou hašovací funkci. Bity, které nejsou použity jako klíč, tedy „odpad“ po zkrácení, mohou být použity jako „sůl“ pro další zpracování. V případě tohoto použití není bezkoliznost funkce až tolik podstatná, resp. prolomení bezkoliznosti hašovací funkce neohrožuje celkový výsledek natolik, aby prolomená funkce nemohla být nadále používána.

⇒ **Ukládání a kontrola hesel**

Dalším použitím je kontrola a ukládání hesel. Uživatel si při registraci do daného systému zvolí (případně je mu přiděleno) uživatelské jméno a heslo. Z těchto hodnot je vypočítán prostřednictvím hašovací funkce otisk, který je uložen do databáze. Výhoda ukládání otisků, namísto konkrétního jména a hesla, je v rychlé kontrole (systém porovná otisk s již existujícími otisky v databázi, čímž zjistí, zda neexistuje jiný uživatel se shodným jménem a heslem, daleko rychleji než kdyby musel porovnávat kombinaci jednotlivých jmen a hesel zvlášť) a zároveň, pokud dojde k narušení databáze, útočník není schopen zpětně identifikovat z otisku jméno a heslo, které uživatelé používají. Tato skutečnost jasně demonstruje potřebu bezkoliznosti hašovací funkce.

⇒ **Tvorba UUID/GUID**

Hašovací funkce jsou využívány pro tvorbu univerzálních jedinečných identifikátorů (UUID) známých také jako globální unikátní identifikátory (GUID). Jelikož otisk získaný prostřednictvím hašovací funkce je teoreticky jedinečný, lze na jeho základě konstruovat jedinečné identifikátory. Konkrétním příkladem takového identifikátoru je URN [107].

⇒ **Integrita a detekce modifikace dat**

Hašovací funkce jsou dále využívány ke kontrole integrity dat. Jelikož hašovací funkce pracuje s binární reprezentací dat (zprávy), změna byť jediného bitu, vede ke změně celého otisku zprávy. Hašovací funkce tedy umožňují, zjisti, zda nebyla zpráva při přenosu poškozena. Pro lepší představu, přenáším-li soubor dat z bodu A do bodu B, je vypočítán otisk souboru před přenosem a otisk souboru po přenosu, jsou-li otisky shodné mám jistotu, že soubor byl přenesen bez poškození.

Detekce modifikace dat je užitečná nejen ve spojení s digitálním podpisem, nýbrž i v běžných aplikacích. Příkladem může být třeba textový editor.

2.6. Shrnutí

Hašovací funkce jsou velmi důmyslným nástrojem moderní kryptologie. Ačkoli se jedná o nástroj poměrně mladý, jeho praktické použití je velice široké a neomezuje se pouze na digitální podpis, pro který bylo toto primitivum původně koncipováno.

Uvést všechny možnosti praktického využití hašovacích funkcí snad ani nelze, jelikož oblastí ve kterých je možné uplatnit „jedinečný“ datových řetězec je teoreticky velmi mnoho. Ze zde uvedených příkladů je patrné, že výstup získaný prostřednictvím hašovacích funkcí nemusí být finálním produktem, ale může sloužit jako vstup pro funkce jiné, přičemž tyto mohou opět využívat hašovací funkce.

Základní teoretický vývoj kryptografických hašovacích funkcí proběhl mezi lety 1976 a 1989. Mezi těmito lety byly vytvořeny definice, které jsou v dnešních dobách dále upřesňovány a rozšiřovány a to zejména v souvislosti s praxí.

Celkově byly vytvořeny tři definice funkcí „jednocestná hašovací funkce“ (OWHF), „bezkolizní hašovací funkce“ (CRHF) a „univerzální jednocestná hašovací funkce“ (UOWHF), přičemž prakticky nejpoužívanější a nejčastěji konstruované jsou právě CRHF.

Obecným problémem je nejednotnost v terminologii a to, jak na poli českém tak mezinárodním. Vzhledem k české terminologii by možná nebylo od věci, stanovit jednotné české ekvivalenty anglických termínů, a to tak, aby nešlo o přímý překlad, jelikož ten v textu může působit poněkud problémy. Proto jsem také v některých částech použila anglické výrazy namísto češtiny.

Samotný výraz „hash function“ nalezneme v české literatuře v různých podobách. Jsou používány termíny „hashovací funkce“, „hešovací fukce“ a „hašovací funkce“. Osobně jsem volila termín poslední a to z toho důvodu, že dle mého názoru je tento nejlepším českým ekvivalentem termínu anglického.

Vzhledem k terminologii bych si dovolila ještě jednu poznámku, a sice vztahující se k tiskové zprávě Ministerstva Vnitra ČR [81], ve které informuje uživatele datových schránek o plánovaném přechodu na SHA-2. V této zprávě je SHA-2 označena jako šifrovací algoritmus, což může být do jisté míry zavádějící, neboť SHA-2 je hašovací funkcí.

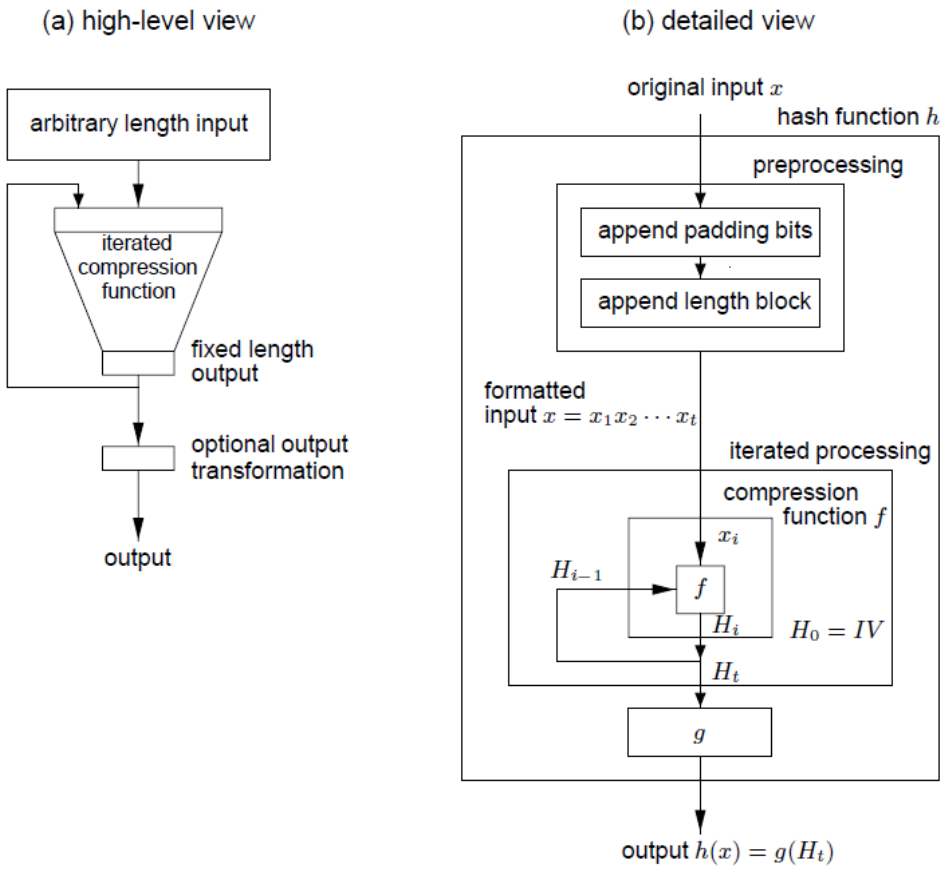
Hašovací funkce v sobě nesou prvky jak symetrické tak asymetrické kryptografie nicméně nejsou šiframi. Šifry, na rozdíl od hašovacích funkcí, nejsou primárně konstruovány jako

noninvertibilní, tj. při znalosti určitého klíče je možno provést zpětné dešifrovací zobrazení, což u kryptografických hašovacích funkcí prvoplánově nelze. Zároveň šifry neposkytují pevnou délku výstupu, resp. po aplikaci šifrovacího klíče na text sice může dojít k redukci délky vlastních dat, ale délka těchto výstupních dat není pevně stanovena.

3. Konstrukce hašovací funkce

Většina současných hašovacích funkcí využívá iterované zpracování, tj. zpracovává zprávu po částech cyklickou aplikací jiné funkce či procedury. Iterované zpracování v obecném pojetí demonstruje následující schéma (Obrázek 3).

Zpráva libovolné konečné délky je rozdělena do bloků x_i, které jsou zpracovávány kompresní funkcí f.



Obrázek 3 Obecný model iterované hašovací funkce [74, s. 332]

Jak je vidět na obrázku, zpráva je nejprve předzpracována doplněním určitým počtem bitů tak, aby ji bylo možno rozdělit do bloků o pevné délce. Poté je zpráva rozdělena do bloků a následuje zpracování kompresní funkcí. Každý blok zprávy je zpracován kompresní funkcí čímž je získána hodnota H_i. Tento mezivýsledek neboli kontext je pak v dalším kroku vstupem kompresní funkce. Funkce f tedy zpracovává dva vstupy, vlastní blok zprávy a kontext. H_0 je iniciační hodnota, tedy hodnota kontextu, při zpracování prvního bloku zprávy.

H_t je výsledná hodnota po zpracování všech bloků x_i, g je potom výsledná transformace H_t často rovná $g(H_t) = H_t$.

Rozdíl mezi jednotlivými konkrétními funkcemi spočívá v tom, jaký postup využívají při předzpracování, volbě iniciačních hodnot, vlastní kompresní funkcí a výslednou transformací zobrazení.

3.1. Merkle-Damgårdova konstrukce hašovací funkce

Metodu iterované hašovací funkce poprvé představili (nezávisle na sobě) Merkle [77] a Damgård [31] na konferenci Crypto v roce 1989. Oba shodně přišli na to, že pro konstrukci funkce *H*, která by zpracovávala libovolně dlouhý vstup je snazší definovat funkci *F* která by zpracovávala vstup o pevné délce, přičemž tato funkce *F* by měla stejné vlastnosti jako funkce *H* (Merkle [77] toto nazývá meta metoda a můžeme ji nalézt již v jeho práci z roku 1979 [75]). Z toho plyne nutnost zarovnání původní zprávy, tak aby ji bylo možné rozdělit do částí, tak velkých jako je definovaná velikost vstupu funkce *F*. Pro možnost jednoznačného odejmutí doplňku zprávy je nutné nějakým způsobem vyjádřit původní délku zprávy. Tomuto doplnění o délku zprávy se říká Merkle-Damgårdovo zesílení.

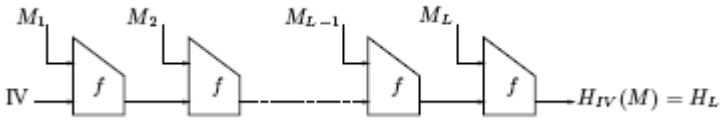
Algoritmus konstrukce je následující:

- ⇒ velikost vstupu funkce $f = (r + j)$ kde j *je* rovno délce výstupu funkce *H*
- ⇒ zpráva x o délce n je zarovnána (je-li to nutné) co nejmenším počtem nulových bitů tak, aby její velikost byla násobkem r a bylo ji možné rozdělit do t bloků o pevné délce r
- ⇒ zpráva je rozdělena do bloků $x_1, x_2 \ldots x_t$
- ⇒ je vytvořen poslední koncový blok x_{t+1}, který je bitovým vyjádřením absolutní hodnoty délky zprávy x před zarovnáním (zde předpokládáme $n < 2^r$) s potřebným počtem nulových bitů jako prefix pro dosažení délky r
- ⇒ potom výsledný otisk získám tímto způsobem: $h(x) = H_{t+1} = f\ (H_t \parallel x_{t+1})$

$$H_0 = 0^n;\ H_i = f(H_{i-1} \parallel x_i);\ 1 \leq i \leq (t + 1)$$

Grafické znázornění celé konstrukce ilustruje níže uvedené schéma (Obrázek 4). Zpráva je zarovnána a rozdělena do bloků M_1 až M_L, přičemž blok M_L je blokem nesoucí délku zprávy. První blok zprávy vstupuje do kompresní funkce (Merkle ve své práci využíval pro kompresní

funkci DES[6]) s předem definovanou iniciační hodnotou jako klíčem. Výstupem zpracování je hodnota, která představuje nový klíč (kontext) pro zpracování dalšího bloku zprávy. Takto se celý proces opakuje, dokud nejsou zpracovány všechny bloky zprávy *M*, přičemž poslední výstup kompresní funkce *f* je otiskem zprávy *M*.



Obrázek 4 Merkle-Damgårdova konstrukce [45, s. 409]

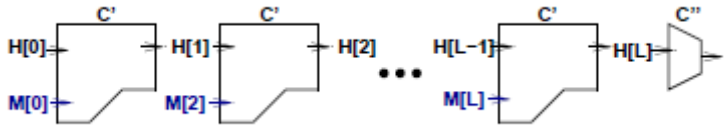
U této konstrukce bylo prokázáno, že pokud kompresní funkce *f* bude bezkolizní, bude bezkolizní také hašovací funkce z ní tímto způsobem vytvořená.

Tato konstrukce se stala základem mnoha současných hašovacích funkcí a prakticky téměř všech dalších publikovaných obecných konstrukcí hašovací funkce.

3.2. Další konstrukce hašovací funkce

V souvislosti s útoky na Merkle-Damgårdovu konstrukci (MD), respektive útoky na funkce, které tuto konstrukci využívaly, vznikla celá řada konstrukcí, které MD modifikují.

V roce 2004 publikoval Lucks [69] dvě konstrukce, a sice „Wide-Pipe Hash“ a „Double-Pipe Hash“. Wide-Pipe Hash na rozdíl od MD konstrukce používá dvě kompresní funkce *C´* a *C´´* (viz. Obrázek 5).



Obrázek 5 Wide-Pipe Hash [69, s. 8]

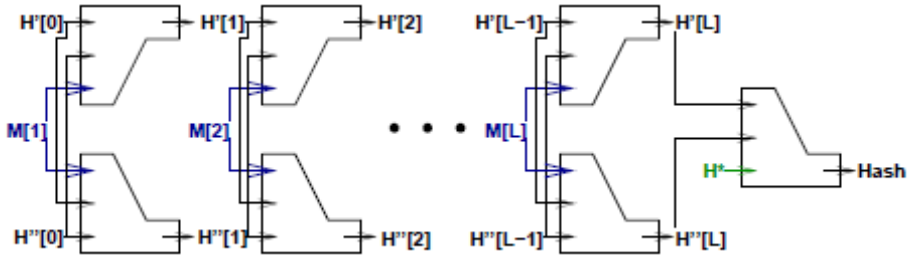
Princip je stejný jako u Merkle-Damgårda až do bodu H_L. H_0 je náhodně vybraná iniciační hodnota. *M* je zpráva rozdělená do M_1 až M_L bloků a je postupně zpracovávána kompresní funkcí *C´*. H_L, tedy výsledek po zpracování všech bloků, označuje Lucks jako „střední haš“, který tvoří vstupní hodnotu do druhé kompresní funkce *C´´*. Výstupem celé funkce je pak

[6] DES (Data Encryption Standard) – Představuje bloková šifra, která byla vytvořena v první polovině 70tých let minulého století. Samotný algoritmus byl vytvořen týmem z IBM na podkladu Feistelovy šifry Lucifer. DES jako standard samotný byl zamýšlen pro ochranu tajných vládních dat. Algoritmus byl navržen pro potřeby tohoto standardu v roce 1974 . V roce 1976 byl schválen a v roce 1977 byl publikován jako standard FIPS PUB 46. V letech 1988 a 1993 byl standard revidován. V roce 1999 byl algoritmus prolomen a v rámci tohoto standardu jej nahradil Triple-DES. V roce 2004 byl standard oficiálně vyjmut ze seznamu standardů.

$H(M) = C''(H_L)$. U této konstrukce je jasně odvoditelný předpoklad, že vstup a kompresní funkce C´ bude daleko větší než vstup funkce C''.

Tato konstrukce vznikla jako reakce na útok, který popsal Joux [70], který umožňuje nalézat multikolize u iterovaných hašovacích funkcích založených na MD. V tomto útoku dokazuje, že složitost nalezení druhého vzoru je u těchto funkcí rovna $log(k) \times 2^{n/2}$, kde k je číslo počítané hašovací hodnoty.

Konstrukce Double-Pipe Hash (Obrázek 6) vypadá poněkud složitěji, jedná se však pouze o zdvojení původní MD.



Obrázek 6 Double-Pipe Hash [69, s. 12]

Konstrukce používá jednu kompresní funkci C a tři různé iniciační hodnoty H', H'', H^*. Blok zprávy M, vstupuje do jedné kompresní funkce dvakrát vždy zřetězený s jinou z iniciačních hodnot (H', H''), přičemž je jasné, že H' se nesmí rovnat H''. Formálně tedy:

$H_i' = C(H_{i-1}', H_{i-1}'' \| M_i)$ a $H_i'' = C(H_{i-1}'', H_{i-1}' \| M_i)$.

V L-tém kroku (tedy po zpracování všech bloků zprávy M) dva výstupy H_L' a H_L'' vstupují znovu do kompresní funkce C spolu s poslední hodnotou H^*. Celý výstup funkce potom tvoří $H(M) = C(H^*, H_L' \| H_L'' \| 0^{m-n})$, kde 0^{m-n} představuje přidání nulových bitů tak, aby byla dosažena požadovaná délka bloku.

Krom těchto, přišli v roce 2004 Gauravaram, Millan a May [44], ještě s jednou konstrukcí a sice CRUSH. Tato konstrukce měla být průlomem, jelikož poskytovala zcela nový pohled na konstrukci hašovacích funkcí. Byla založena na technice iterovaného dělení.

Jak však bylo dokázáno [4, 53], navrhovaná konstrukce se pro konstrukci hašovacích funkcí ukázala být nevhodnou, jelikož vykazuje minimální odolnost vůči útokům. Dokonce, jak uvádí [4], i v silnější verzi vyžaduje nalezení kolize menší časovou komplexitu než u narozeninového útoku.

V roce 2006 byla prezentována konstrukce HAIFA [18] (HAsh Iterative FrAmevork). Tato konstrukce přidává na vstup kompresní funkce, oproti MD, další dva parametry, a sice „sůl“ a

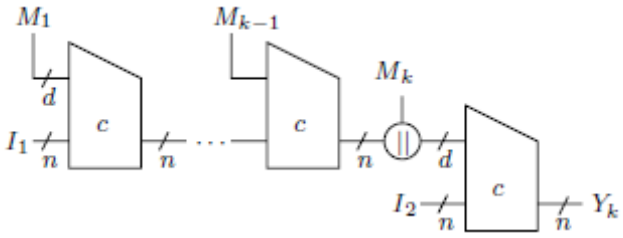
určitý bitový řetězec „*#bits*“. Řetězec označený jako „*#bits*“ představuje počet bitů, který byl již zpracován. Volba soli je ponechána na konkrétní aplikaci, solí může být například náhodný řetězec, sériové číslo aplikace nebo řetězec identifikující aplikaci.

Zpráva je zarovnána podobně jako u MD. Nejprve je připojen jeden 1 bit a po té potřebný počet nulových bitů, tak aby platilo *mod n = (n – (t + r)*, kde *t* je délka zprávy v bitech a *r* je délka výstupu hašovací funkce v bitech.

HAIFA, krom zesílení zprávy (vyjádření délky zprávy), přidává ještě k zarovnání délku výstupu funkce v (délku otisku). Iniciační hodnota může být libovolná dle délky požadovaného výstupu kompresní funkce a je odvozena takto: $IV_m = C(IV, m, 0, 0)$.

IV je libovolná iniciační hodnota, *m* je požadovaná délka výstupu funkce a C kompresní funkce. Samotné zpracování pak představuje: $h_i = C(h_{i-1}, M_i, \#bits, salt)$. Po zpracování může být celý výstup kompresní funkce ještě zkrácen, tedy za předpokladu, že délka výstupu kompresní funkce není rovna požadované délce výstupu hašovací funkce.

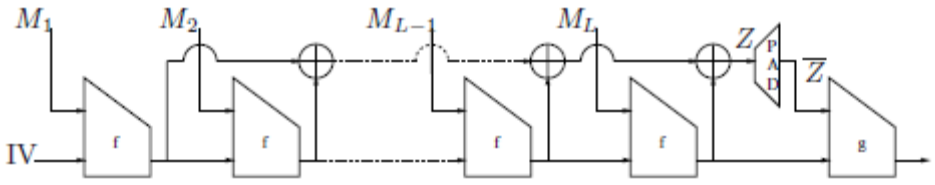
Další konstrukcí, publikovanou v roce 2006 [7], je EMD (Enveloped Merkle-Damgård). Grafické znázornění této konstrukce není, dle mého názoru, sice nejlepší nicméně je převzato tak, jak je uvedeno v originálu dokumentu (viz.Obrázek 7).



Obrázek 7 Konstrukce EMD [7, s. 311]

Princip této konstrukce spočívá v tom, že jsou dány dvě fixní iniciační hodnoty, pro které platí, že $IV_1 \neq IV_2$. Zpráva je rozdělena do bloků o délce $d \geq n + 64$. Blok M_k je posledním blokem, který je 64 bitovým vyjádřím délky zprávy *M*. Poslední blok je tedy menší než všechny ostatní. V k-tém kroku, dochází k zřetězení hodnot M_k a kontextu I_1 a do kompresní funkce vstupuje tato hodnota spolu s druhou iniciační hodnotou $IV_2 = I_2$.

Krom HAIFA a EMD byly v roce 2006 publikovány ještě konstrukce 3C, 3CWP a 3C+ [45]. Konstrukce 3C (Obrázek 8) používá dvě kompresní funkce *(f, g)* a dvě kontextové hodnoty *u* a *w*. Algoritmus, který konstrukce používá je následující: Pro $i = 1$ až L, $w_i = f(w_{i-1}, M_i)$, kde $w_0 = IV$ *a* $u_1 = w_1$. Potom pro $i = 2$ až L, $u_i = u_{i-1} \oplus w_i$.

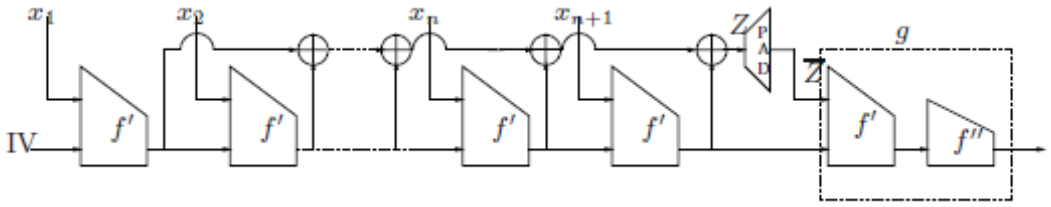


Obrázek 8 Konstrukce 3C [45, s. 412]

Jasněji řečeno začátek je stejný jako u MD. Zpráva je zarovnána (je-li to nutné) nulami a rozdělena do bloků. Poslední blok nese délku zprávy *M*. V prvním kroku do kompresní funkce *f* vstupuje blok zprávy a iniciační hodnota $IV = w_0$. Po prvním kroku je nastavena hodnota druhého kontextového řetězce u_1 na stejnou hodnotu jako w_1 . Po druhém kroku je před vstupem do kompresní funkce provedena XOR operace na horním „akumulačním“ řetězci u_i. Takto pokračuje 3C až do zpracování celé zprávy. Po zpracování všech bloků je výsledné $u_L = Z$ ještě nějak zarovnáno (na obrázku vyznačeno PAD), je-li to třeba.

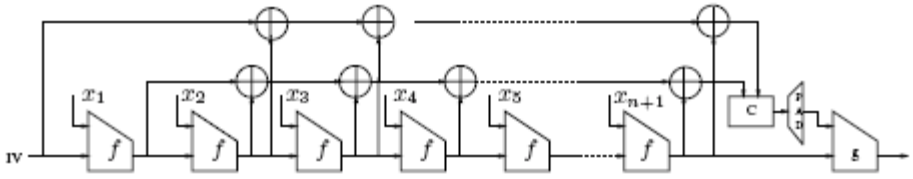
V popisu konstrukce není specifikováno, jak by mělo zarovnání probíhat. Pravděpodobně je tedy přidán potřebný počet nulových bitů jako prefix tak, aby celková délka *Z* + délka w_L odpovídala délce vstupu kompresní funkce *g*. Takto získané $\overline{Z}$ (tedy *Z* po zarovnání) potom vstupuje s w_L do druhé kompresní funkce *g*. Výsledné zobrazení tedy tvoří g($\overline{Z}$, w_L).

Hybridní variací této konstrukce je verze „wide-pipe“ tedy C3WP (Obrázek 9). Proces hašování je stejný jako výše popsaný. Jediný rozdíl je v tom, že *g* je nahrazeno opětovným použitím *f* a dalším použitím funkce *f´´*.



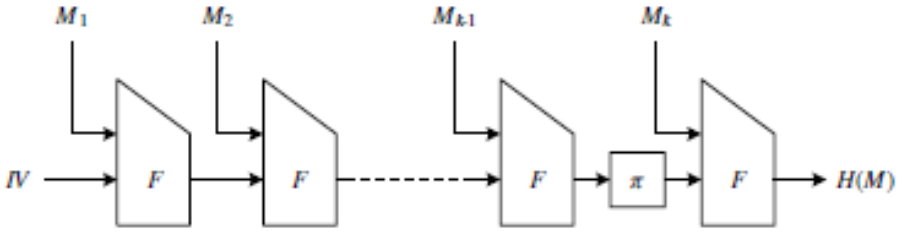
Obrázek 9 Konstrukce 3CWP [45, s. 417]

Poslední z řady 3C konstrukcí je 3C+. Oním „plus“ je zde přidání ještě jednoho akumulačního kontextu. Princip je opět stejný jako u 3C. V bodě *c* se akumulační kontexty zřetězí a po zarovnání vstupují do kompresní funkce *g* spolu se spodním řetězcovým kontextem *w* (tedy výsledkem po zhašování všech bloků).



Obrázek 10 Konstrukce 3C+ [45, s. 418]

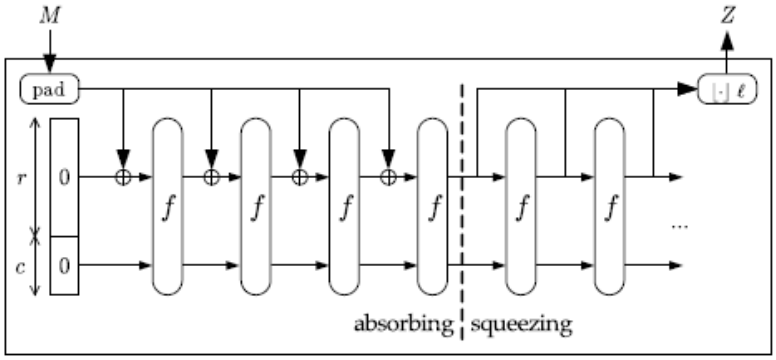
Další konstrukci představili Hirose, Park a Yun [54] v roce 2007. Konstrukce MDP (Merkle-Damgård with Permutation, Obrázek 11) je, jak název napovídá, MD konstrukce s přidanou permutací po předposledním kroku. Permutaci autoři předpokládají jako fixně stanovený parametr v dokumentaci hašovací funkce.



Obrázek 11 Konstrukce MDP [54, s. 120]

Jiná konstrukce představená v roce 2007 je Sponge [12]. Sponge konstrukce (Obrázek 12) je založena na „sponge“ funkci. Funkce sponge má tři parametry *f*, *pad* a *r*. Symbol *f* značí transformaci (permutaci) fixní délky *b*, „*pad*“ je pravidlo pro zarovnání a *r* je parametr nazývaný „bitrate“. Délka vnitřního stavu je dána $b = r + c$, kde je *c* kapacita. Nejprve je vnitřní stav iniciován a nastaven na hodnotu 0. Zpráva je zarovnána a rozdělena do *r* bloků.

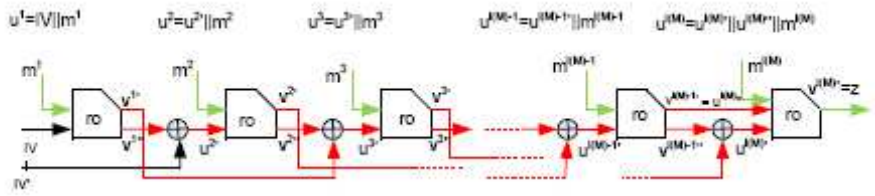
Zpracování je rozděleno do dvou částí, „absorbace“ a „vymačkání“ (tyto termíny představují analogii s houbou; anglicky „sponge“). V první části je provedena XOR operace mezi r-bitovým blokem a první r-bitovým vnitřním stavem a výsledek je vložen do funkce *f*, kde je provedena permutace. Po zpracování všech bloků, konstrukce přepne do fáze druhé. V této fázi vrací konstrukce prvních *r* bitů jako výstupní blok zpracovaných *f*. Počet iterací je určen požadovanou délkou výstupu *l*. Ve výsledku je výstup zkrácen na prvních *l* bitů.



Obrázek 12 Konstrukce Sponge [12, s. 13]

Vnitřní stav *c* není nikdy přímo dotčen vstupem a výstupem bloků a určuje dosažitelnou úroveň bezpečnosti konstrukce.

Posledním příkladem konstrukcí je FWP (Fast Wide-Pipe, Obrázek 13) z roku 2010 [80].



Obrázek 13 Konstrukce FWP [80, s. 3]

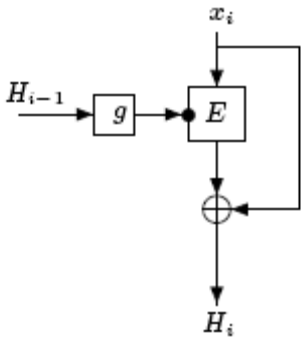
U této konstrukce je nejprve zpráva *M* zarovnána tak, že je přidáno *t* nulových bitů a 64-bitové vyjádření délky zprávy *M*. Číslo *t* získám výpočtem $|M| + t + n + 64 = 0 \bmod l$ pro $t \geq 0$, kde *n* je délka výstupu funkce a *l* je délka bloku zprávy. Pak je zpráva rozdělena do bloků o délce *l*, přičemž poslední blok nesoucí délku zprávy má velikost $l - n$. Dále máme 2 iniciační hodnoty h_{i-1} a $h'_{i-1} = 0^n$. Blok zprávy vstupuje do kompresní funkce spolu s iniciační hodnotou $IV = h_{i-1}$. Výstupem je hodnota $C(h_{i-1} \| M_i) = h''_i \| h'_i$ poté je provedena XOR operace $h_i = h''_i \oplus h'_{i-1}$, proces se opakuje až do k-2 kroku, kdy výstupem funkce je hodnota h_{k-2}, h'_{k-2}. Poté je zpracován poslední blok zprávy $C(h_{k-2} \| h'_{k-2} \| M_{k-1})$. Výstupem celé funkce je pak $h = h'_{k-1}$.

3.3. Konstrukce kompresní funkce

Jak ukazuje MD konstrukce je bezpečnost, resp. bezkoliznost, hašovací funkce dána právě kompresní funkcí. Proto jako příklad tři nejznámější konstrukce kompresní funkce pro hašovací funkce založené na blokových šifrách.

Konstrukce Matyas-Meyer-Oseas byla publikována v roce 1985 [71]. Na grafickém znázornění konstrukce (Obrázek 14), vidíme, že blok zprávy x_i vstupuje do *E*, což je bloková

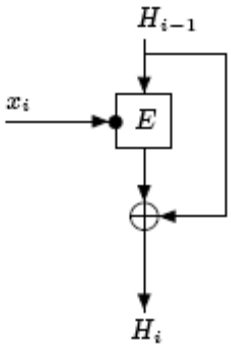
šifra, spolu s hodnotou H_{i-1}, která představuje klíč k této šifře. Uvnitř E je text x_i zašifrován pomocí klíče H_{i-1}. Pokud délka klíče není adekvátní (např. mají-li bloky zprávy různou velikost) je uvnitř g provedena potřebná transformace (zarovnání). Po zašifrování je provedena XOR operace nad zašifrovaným a prostým x_i, čímž získám novou hodnotu klíče H_i.



Obrázek 14 Konstrukce Matyas-Meyer-Oseas [74, s. 340]

Matematický zápis funkce vypadá tedy takto: $H_0 = IV;\ H_i = E_{g(Hi-1)}(x_i) \oplus x_i;\ 1 \leq i \leq t$
kde t je počet bloků zprávy x a předem H_0 definovaná hodnota [74, s. 341].

Druhá konstrukce (Obrázek 15) je přisuzována autorům D. Daviesovi a C. Meyereovi, proto je nazývána „konstrukce Davies-Meyer“ [99, 93].

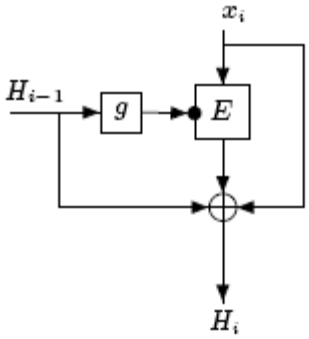


Obrázek 15 Konstrukce Davies-Meyer [74, s. 340]

Do blokové šifry E vstupuje blok zprávy x_i a klíč H_{i-1}. Na výstupu je provedena XOR operace nad H_{i-1} a H_{i-1}. Formální zápis funkce zní: $H_0 = IV;\ H_i = E_{xi}(H_{i-1}) \oplus H_{i-1};\ 1 \leq i \leq t$ [74, s. 341]. Rozdíl proti předchozí konstrukci je v tom, že délka klíče není závislá na délce bloku zprávy.

Poslední konstrukce (Obrázek 16) byla navržena nezávisle Preneelem [93] a kolektivem Miyaguchi, Otha, a Iwata [79]. Formální zápis funkce je následující:

$H_0 = IV; H_i = E_{g\,(Hi-1)}(x_i) \oplus x_i \oplus H_{i-1}; 1 \leq i \leq t$ [74, s. 341]. Algoritmus funkce v sobě nese skloubení dvou předchozích funkcí. Do funkce vstupuje blok zprávy a klíč. Na výstupu je provedena XOR operace nad prostým textem, zašifrovaným textem a klíčem, čímž získáme nový klíč H_i.



Obrázek 16 Konstrukce Miaguchi-Perneel [74, s. 340]

Všechny uvedené funkce mají v rámci hodnocení efektivity hašovacích funkcí (resp. funkcích založených na těchto kompresních funkcích) hodnotu 1. Efektivita je hodnocena jako poměr počtu operací potřebných ke zpracování jednoho bloku zprávy.

Tyto funkce patří mezi dvanáct prokazatelně bezpečných schémat používajících noninvertiblilní řetězení [94].

3.4. Shrnutí

Merkle-Damgårdova konstrukce hašovací funkce je dodnes nejpoužívanějším teoretickým podkladem pro konstrukci hašovacích funkcí. Tuto skutečnost dokazují i variace na tuto konstrukci uvedené v této kapitole. Hlavním motivem všech těchto dalších konstrukcí byly zejména dva generické útoky proti Merkle-Damgård. Prvním byl multikolizní útok Jouxe z roku 2004 [58] a druhým útok Kelsey a Schneiera z roku 2005 [60].

Teoretických konstrukcí hašovacích funkcí existuje pravděpodobně více, než je zde uváděno. Žádná z nich však zatím nedosáhla takového rozšíření na poli teoretickém i praktickém jako právě konstrukce Merkle-Damgård.

To samé platí i pro výše uvedené teoretické konstrukce kompresních funkcí. Opět, teoretických konstrukcí kompresních funkcí existuje mnoho, samotný Preneel jich ve své disertační práci [93] navrhl hned několik. V analýze [94], kterou vytvořil spolu s Govaertsem

a Vandewallem v roce 1993, bylo hodnoceno celkem 64 modelů a jako prokazatelně bezpečných bylo určeno dvanáct.

Zde bych chtěla podotknout, že existují ještě konstrukce s hodnocením např. 1/2, jenž zpracovávají blok zprávy ve dvou operacích, případně 1/4. Takovéto konstrukce jsou využívány zejména u klíčovaných funkcí.

4. Současné hašovací funkce

V předchozích kapitolách byly popsány teoretické základy a podklady pro tvorbu hašovacích funkcí. V této kapitole bude věnována pozornost již konkrétním hašovacím funkcím, které vznikaly v období od roku 1989 do roku 2008.

Zvláštní pozornost bude věnována zejména funkcím z rodiny MDx, které se staly základnou pro mnoho funkcí vzniklých především v devadesátých letech minulého století. Dále pak funkcím SHA, které jsou definovány jako bezpečné hašovací algoritmy a společně s funkcemi MDx se řadí mezi celosvětově nejrozšířenější.

4.1. Stručný přehled

Jednou z prvních prakticky realizovaných hašovací funkcí byla funkce MD2. Funkce byla vytvořena v roce 1989 a jejím autorem byl Ronald Rivest. Blíže se této funkci, stejně jako ostatním z rodiny MDx, budu věnovat v samostatné podkapitole.

V dalším roce vznikly hned 4 hašovacích funkce, a sice GOST, MD4, N-Hash a Snefru.

Tvůrcem GOST byla ruská Federální agentura pro vládní komunikaci a informace a byla definována v standardu GOST 34.11-94 [107]. Jedná se o iterovanou hašovací funkci poskytující délku výstupu 256 bitů, která je založena na blokové šifře GOST, jenž je ruskou variací DES. V roce 2008 byl publikován útok, který umožňuje nalezení kolize se složitostí 2^{105}, tuto funkci tedy považujeme za prolomenou [72].

Tvůrci N-Hash byli Miyaguchi, Ohta a Iwata [79]. Funkce poskytuje délku výstupu 128 bitů a je založená na blokové šifře FEAL-N. Útok na tuto funkci publikovaný hned následující rok ukázal, že nalezení kolidujícího páru můžeme realizovat na, v té době, běžném počítači během zhruba během dvou hodin [1].

Funkce Snefru byla navržena jako silná hašovací funkce na zakázku pro Xerox. Jejím tvůrcem byl Merkle. Funkce podporuje výstup o 128 a 256 bitech. Bezpečnost funkce je dána kompresní funkcí, resp. počtem průchodů kompresní funkcí. Každý průchod přestavuje 64 kol zpracování. Počet průchodů může být 2, 3, 4 a 8. Jak uvádí Biham a Shamir [17], Snefru s dvěma průchody je snadno prolomitelná. S využitím běžného počítače trvá prolomení 3 minuty a nelezení druhého vzoru zhruba hodinu. Po publikování útoku byla vytvořena osmi průchodová verze. Ta jako jediná zůstává bezpečná [19].

V roce 1991 byly představeny dvě nové funkce CellHash a FFT-Hash. FFT-Hash byla funkce založená na rychlé Fournierově transformaci (Fast Fourniere Transformation). Funkce byla

neformálně představena na konferenci Crypto v roce 1991 a jejím autorem byl C. P. Schnorr. O rok později byla již formálně představena její vylepšená verze FFT-Hash II [120]. Z toho také vzniklo pozdější dodatečné označení původní verze FFT-Hash I. Tato původní verze se ukázala, jako kolizní ještě ten samý den kdy byla představena. Proto přišel Schnorr s vylepšením funkce, které jí přidalo pořadové označení II (resp. I pro původní verzi). Ještě téhož roku byl publikován článek [123], který ukazuje, že ani vylepšená verze FFT-Hash II není bezkolizní. V roce 1993 byla představena verze Parallel FFT-Hash [121], která, jak ukazuje [37], není příliš odolná vůči nalezení vzoru.

Funkce CellHash byla založena na modelu celulárního automatu a její kompresní funkce byla inspirována kompresní funkcí MD4. Tvůrci CellHash byli Daemen, Vanderwale a Govaerts. Funkce byla přihlášena do projektu RIPE[7], avšak byla zamítnuta (duben 1992), jelikož její odolnost vůči kolizím byla shledána jako slabá [30]. Tvůrci přišili v roce 1992 ještě s dalšími funkcemi, upravenou verzí CellHash, SubHash a Boognish. V roce 2006 byl publikován útok proti CellHash a SubHash, který poukazoval snadno invertibilní vnitřní strukturu a předkládal doporučení pro zesílení obou funkcí [36].

V roce 1992 byla vytvořena, v rámci projektu RIPE, funkce RIPEMD. RIPEMD byla první z pěti hašovacích funkcí, které byly v rámci tohoto projektu vytvořeny. RIPEMD poskytovala výstup o 128 bitech a měla být silnější variantou funkce MD4, jak se však ukázalo [35], funkce nebyla bezkolizní, proto v roce 1996 vznikly další variace této funkce RIPEMD-128, RIPEMD-160, RIPEMD-256 a RIPEMD-320 [24].

Rok 1992 přinesl také funkci HAVAL [133]. HAVAL je funkcí poskytující libovolnou délku výstupu (Hash Algorithm with VAriable Length of output), resp. výstup o délce 128, 160, 192, 224 nebo 256 bitů. Funkce zpracovává bloky o délce 1024 bitů. Uživatel má možnost výběru počtu průchodů kompresní funkcí (podobně jako u Snefru). Průchodů může být 3-5. Jak tvůrci uvádí, se třemi průchody je HAVAL o 65% rychlejší než MD5. Proti funkci bylo publikováno mnoho útoků, avšak až v roce 2012 byl publikován první útok na kompletní pěti-průchodovou verzi. Útok je prozatím prakticky neuskutečnitelný [113].

V roce 1996 byla publikovaná funkce Tiger [3], která byla, jako patrně první, primárně navržena pro 64-bitové procesory. Funkce poskytuje možnost výstupu 192, 160 a 128 bitů. Tiger zpracovává zprávu po blocích o velikosti 512 bitů. V základní verzi užívá Tiger tři

[7] Projekt RIPE [23] – Projekt v rámci programu EEC-RACE zaměřený na vytvoření bezpečných kryptografických primitiv. V rámci projektu vznikly, krom RIPEMD, také funkce RIPEMAC a IBC-hash koncipované pro tvorbu MAC kódů.

průchody kompresní funkcí. Funkce byla zaslána jako kandidát do projektu NESSIE[8] (New European Schemes for Signatures, Integrity And Encryption, 2000-2003), nicméně do portfolia kryptografických primitiv vybrána nebyla. Existuje také její nepublikovaná druhá verze. Funkce Tiger prozatím nebyla zcela prolomena.

V roce 1998 vznikla funkce Panama [28]. Tvůrcem byli Daemen a Clapp. Panama byla založena na funkci „StepRightUp“, kterou publikoval Daemen v rámci své disertační práce v roce 1995. Panama byla využitelná nejen jako hašovací funkce, ale také jako proudová šifra. Jako hašovací funkce poskytuje výstup o 256 bitech. Hašovací funkce byla prolomena v roce 2001, na základě útoku umožňující nalezení kolize se složitostí 2^{82}. V roce 2007 byl tento útok ještě vylepšen a složitost nalezení kolize byla snížena na 2^{6} [29].

V roce 1998 byla publikována také PKC-HASH [114]. Funkce byla založená na rodině funkcí MDx, zpracovává zprávu po blocích o velikosti 512 bitů a poskytuje výstup o velikosti 160 bitů. Tato funkce byla speciálně navržena pro konstrukci autentizačních kódů MAC. V roce 2002 byla funkce oficiálně prolomena na základě publikovaného útoku [50], který umožňuje nelezení kolize se složitostí 2^{30}.

Rok 2000 přinesl celou řadu hašovacích funkcí, krom přírůstků do rodiny funkcí SHA také funkce HAS-160, HAS-V a Whirlpool. Funkce HAS-160 byla navržena pro účely korejského standardu pro digitální podpis a definována byla jako TTAS.KO-12.0011/R1. Funkce je založena z části na funkci SHA-1 a HAS-V. V roce 2007 byl publikován útok umožňující nalezení kolize po 53 krocích s komplexností 2^{35} hašovacích výpočtů [73]. Funkce HAS-V [90] je navržena jako funkce poskytující variabilní délku výstupu od 128-320 bitů.

Funkce Whirlpool byla vytvořena Vincentem Rijmenem a Paulo S. L. M. Barretem. Funkce poskytuje výstup o velikosti 512 bitů a je založena na Merkle-Damgårdově konstrukci a Miyaguchi-Preneel konstrukci kompresní funkce. Původní verze Whirlpool-0 byla zaslána do projektu NESSIE. Její vylepšená verze Whirlpool-T byla vybrána do portfolia kryptografických primitiv a po odstranění chyby publikované v [115] byla pod prostým názvem Whirlpool definována jako standardní hašovací algoritmus v rámci ISO/IEC 10118-3:2004 [5, 56].

V roce 2005 byla vytvořena funkce VSH (Very Smooth Hash) [27]. Tato funkce je založena na faktorizaci hladkých čísel a podobá se zpracováním RSA. Velmi hladké číslo představuje *b*

[8] NESSIE [96] – Evropský projekt. Cílem bylo definovat bezpečná kryptografická primitiva. Portfolio primitiv tvořili: blokové šifry, algoritmy pro šifrování s veřejným klíčem, MAC algoritmy a hašovací funkce, algoritmy pro digitální podpis a identifikační schémata. Projekt byl koncipován jako veřejná soutěž. Vybrané algoritmy byly definovány v rámci standardů ISO.

modulo n, pokud největší prvočíslo faktorizace *b* je rovno nanejvýš $(\log n)^c$ a existuje-li číslo *x* pro které platí: $b \equiv x^2 \bmod n$.

V tomto roce byl také konán první kryptografický workshop pořádaný NIST, na němž byly představeny dvě funkce, FORK-256 a DHA-256. Funkce DHA-256 (Double Hash Algorithm [57]) je variací na funkci SHA-256. DHA-256 používá shodné iniciační hodnoty a velmi podobné operace, avšak měla by být, podle autorů, odolnější vůči současným útokům. Na rozdíl od SHA-256 je v DHA-256 slovo zprávy generované při expanzi v rámci jednoho kroku použito dvakrát (proto double).

Funkce FORK-256 [55] používá čtyři paralelní větve zpracování, přičemž v každé větvi je blok zpracováván v osmi krocích. Používá dvě nelineární funkce, které využívají sčítaní modulo 2^{32}, exkluzivní disjunkci a levostranný rotační posun. Zpráva je zpracovávána v blocích o 512 bitech a každý blok tvoří 16 slov. Funkce používá 16 pevně definovaných přidaných konstant. Funkce má předem definovaný pořádek pro vstup slov a konstant do větví. Vstup konstant a slov je různý pro každou větev. FORK-256 by podle autorů měla být o 30% výkonnější (ve smyslu rychlosti zpracování) než SHA-256 a zároveň by měla být odolná vůči současným útokům. V současné době není na tuto funkci znám žádný úspěšný útok.

V roce 2006 byly vytvořeny funkce RadioGatún, LASH, a Edon-R. Všechny tři funkce byly představeny na druhém kryptografickém workshopu pořádaném NIST.

Funkce RadioGatún [11] je další z funkcí na které se podílel Daemen – ten krom již jiných výše uvedených vytvořil v roce 2005 funkci SMASH. SMASH byla prolomena, krátce po své prezentaci na konferenci FSE (Fast Software Encryption) 2005. Spojitost mezi SMASH a RadioGatún spočívá v naprostém odklonění od Merkle-Damgårdovy konstrukce. Funkce RadioGatún staví na funkci Panama a využívá generické kryprografické primitivum nazývané IMF (Iterative Mangling Function). Funkce vrací nekonečně dlouhý řetězec, z nějž je potřebný počet bitů pro výstup hašovací funkce extrahován oříznutím prvních *n* bitů. Zatím není znám žádný útok na tuto konstrukci.

Funkce LASH (Linear Algebra based Secure Hash, LAttice based Secure Hash, Light-weight Aritmetical Secure Hash – mnoho akronymů jenž mohou popsat funkci LASH) [9] byla navržena ve variantách 160, 256, 384 a 512 bitů výstupu a je založena na Merkle-Damgårdově konstrukci a Miyaguchi-Preneel konstrukci kompresní funkce. Vstup kompresní funkce tvoří 640, 1024, 1536 a 2048 bitů, přičemž velikost bloků je rovna $m = n/16$, kde *n* je délka vstupu kompresní funkce. Tedy konkrétně 40, 64, 96 nebo 128 bitů. Funkce používá

pseudo-náhodnou sekvenci založenou na Pollardově generátoru[9]. Kompresní funkce používá na sčítání modulo 256 a exkluzivní disjunkci. V roce 2008 byla publikována kryptoanalýza [120], jenž předpokládá nalezení kolize pro LASH-n v počtu $2^{(4n/11)}$ a nalezení vzoru v počtu $2^{(4n/7)}$ operací. Tento útok využívá jako podklad heuristickou mřížku a poukazuje na nevhodnou volbu iniciační hodnoty, která je tvořena pouze nulovými bity.

Edon-R [48] byla navržena jako nekonečná rodina kryptografických hašovacích funkcí. Jedná se o třídu funkcí s variabilní délkou výstupu. Je založená na kvazigrupách a kvazigrupové řetězcové transformaci. Používá definovanou jednocestnou kvazigrupovou funkci R_1 . Vstup funkce tvoří beztvará kvazigrupa $(Q, *)$ v řádu 2^w (pro $w \geq 4$), číslo N takové, jehož délka je rovna délce výstupu funkce $w \times N$ a zpráva M. Výstupem je hašovací kód o délce $w \times N$ bitů. Zpráva je zarovnána tak, aby její absolutní hodnota po zarovnání byla rovna $k \times N$, kde k je počet kroků. Řetězec o délce $2N$ představující řetězící hodnoty je rozdělen do částí:

$H = (h_0, h_1,\ldots,h_{2N-1})$.

Následuje iniciace počátečních hodnot $H_0 = (0 \bmod 2^w, 1 \bmod 2^w, \ldots 2N - 1 \bmod 2^w)$.

Vlastní zpracovaní je definováno, pro $i = 1$ do k, jako $H_i = R_1(H_{i-1} \| M_i) \bmod 2^{2wN}$

výstupem je potom $Edon\text{-}R(M) = H_k \bmod 2^{wN}$. Tato funkce byla později v upraveném provedení přihlášena do soutěže NIST (viz. kapitola 5), avšak skončila již v prvním kole.

V roce 2007 vznikla funkce Grindahl [65]. Konkrétněji Grindahl-256 a Grindahl-512. Tato funkce je částečně založena na hašovací funkci Snefru a blokové šifře Rijndael. Využívá, jak autoři nazývají, „Concatenate-Permute-Truncate“ tedy zřetězené permutované krácení.

Zpráva je vhodně zarovnána do t bloků o délce $d = d_1 \| d_2 \| \ldots \| d_t$, kde absolutní hodnota d_i je rovna délce zprávy b. Potom pro $0 < i \leq t$ je provedeno:

zřetězení	$S_i \leftarrow d_i \| s_{i-1}$
permutování	$\hat{S}_i \leftarrow P(S_i)$
zkrácení	$s_i \leftarrow trunc_m(\hat{S}_i)$

Absolutní hodnota s_0 je rovna m, kde m představuje stav délky, přičemž platí $m \geq n$ a zároveň $b > 0$. Malé n je délka výstupu funkce. Verze Grindahl-256 byla prolomena ještě téhož roku, útokem [91] vyžadujícím přibližně 2^{112} hašovacích výpočtů k nalezení kolize. Pro verzi Grindahl-512 prozatím nejsou známy žádné útoky.

[9] Pollardův generátor představuje sekvenci u_n pseudonáhodných čísel definovaných vztahem $u_{n+1} \equiv u_n^2 + c \bmod p$.

4.2. Rodina funkcí MDx

Hlavním tvůrcem rodiny hašovacích funkcí MDx je profesor MIT Ronald L. Rivest. Jedná se o sérii algoritmů hašovacích funkcí, kde MD představuje zkráceně „ Message Digest“ (otisk, haš) a připojené číslo, které značí pořadové číslo algoritmu.

Původní funkce MD, někdy také označována MD1 [93], byla proprietárním algoritmem a velmi rychle byla nahrazena funkcí MD2. Funkce MD2 byla vytvořena v roce 1989 a jako RFC 1319 [102] byla publikována v roce 1992. MD2 byla koncipována pro osmi-bitové platformy a poskytuje výstup o velikosti 128 bitů. Algoritmus funkce představují tři kroky. V prvním kroku je provedeno zarovnání zprávy nulovými bity, tak aby délka zprávy byla rovna násobku šestnácti bytů. V případě, že zpráva před zarovnáním odpovídá tomuto násobku, tak je zarovnána přidáním 128 nulových bitů (=16 bytů). Je tedy přidán celý nový blok.

Celkově je tedy ke zprávě přidáno min. 8 až 128 nulových bitů. V druhém kroku je vypočítán 16-bytový kontrolní součet zprávy, který je připojen ke zprávě jako poslední blok. V posledním kroku je zpráva po blocích zpracována kompresní funkcí. Kompresní funkce pracuje se 48-bytovým bufferem *X* a 256-bytovou permutací *S*.

Po dokončení MD2 začal Rivest pracovat na MD3, avšak tato konstrukce se ukázala jako selhání, a proto nebyla nikdy publikována [59, s. 173]. V roce 1990 přišel Rivest s konstrukcí MD4 [109], kterou představil na konferenci Crypto. MD4 již byla koncipována pro 32-bitové platformy a taktéž byla v roce 1992 publikovaná jako RFC standard s číslem 1320 [103].

MD4 se stala základem mnoha dalších konstrukcí (např. RIPEMD, HAVAL, SHA-1 aj.) a je patrně jednou z celosvětově nejznámějších hašovacích funkcí.

Algoritmus funkce probíhá v pěti krocích. V prvním kroku je zpráva zarovnána tak, aby její délka byla rovna 448 modulo 512. Na konec zprávy připojen jeden 1 bit, a poté co nejmenší počet nulových bitů, tak aby bylo dosaženo délky 448 modulo 512. V druhém kroku je připojena délka zprávy. Zbývajících 64 bitů tvoří vyjádření délky zprávy. Délka zprávy je vyjádřena prostřednictvím dvou 32-bitových slov.

Takto předzpracovaná zpráva tedy představuje *x* 512-bitových bloků *m*, přičemž každý blok představuje 16 32-bitových slov.

Ve třetím kroku následuje iniciace bufferu. Iniciační kontext je tvořen čtyřmi 32-bitovými slovy *A*, *B*, *C* a *D*. Pro zpracování zprávy jsou dále definovány následující konstanty a operace:

Hodnoty iniciačních slov:

A = 01234567, B = 89abcdef, C = fedcba98, D = 76543210

Pomocné funkce:

f (X, Y, Z) = $(X \wedge Y) \vee (\neg X)\, Z$

g (X, Y, Z) = $(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$

h (X, Y, Z) = $X \oplus Y \oplus Z$

Přidané „magické“ konstanty:

pro první kolo: y[j] = 0, pro $0 \leq j \leq 15$

pro druhé kolo: y[j] = 5a827999, pro $16 \leq j \leq 31$ (konstanta = $\sqrt{2}$)

pro třetí kolo: y[j] = 6ed9eba1, pro $32 \leq j \leq 47$ (konstanta = $\sqrt{3}$)

Definovaný pořádek pro vstup slov:

z[j] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], pro $0 \leq j \leq 15$

z[j] = [0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15], pro $16 \leq j \leq 31$

z[j] = [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15], pro $32 \leq j \leq 47$

Definovaný počet bitů pro levostrannou rotaci:

s[j] = [3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19, 3, 7, 11, 19], pro $0 \leq j \leq 15$

s[j] = [3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13, 3, 5, 9, 13], pro $16 \leq j \leq 31$

s[j] = [3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15, 3, 9, 11, 15], pro $32 \leq j \leq 47$

Následuje vlastní zpravování zprávy. Blok zprávy, tvořící 16 slov je kopírován do bufferu a je zpracován ve třech kolech o 16 krocích:

(*initialize working variables*) (A, B, C, D) ← (H_1, H_2, H_3, H_4)

(*Round 1*) For j from 0 to 15 do the following:

t ← (A + f(B, C, D) + m[z[j]] + y[j]), (A, B, C, D) ← (D, t <<<s[j], B, C)

(*Round 2*) For j from 16 to 31 do the following:

t ← (A + g(B, C, D) + m[z[j]] + y[j]), (A, B, C, D) ← (D, t <<< s[j], B, C)

(*Round 3*) For j from 32 to 47 do the following:

t ← (A + h(B, C, D) + m[z[j]] + y[j]), (A, B, C, D) ← (D, t <<<s[j], B, C)

(*update chaining values*) (H_1, H_2, H_3, H_4) ← (H_1 + A, H_2 + B, H_3 + C, H_4 + D)

Po zpracování všech bloků tvoří finální výstup funkce zřetězení $H_1 \| H_2 \| H_3 \| H_4$ [109; 74, s. 348]. Rivest ve své práci nabízí ještě možnost rozšíření, pro dosažení 256-bitového výstupu. Jedná se o paralelní zapojení dvou funkcí MD4 nad vstupem. První funkce pracuje výše popsaným způsobem. Druhá funkce pracuje s jinými iniciačními hodnotami slov a přidanou konstantu pro druhé a třetí kolo již netvoří druhá, ale třetí odmocnina ze dvou a tří. Definované konstanty tedy jsou:

Hodnoty iniciačních slov:

A = 00112233, B = 44556677, C = 8899aabb, D = ccddeeff

Přidané konstanty:

pro druhé kolo: y[j] = 50a28be6, pro $16 \leq j \leq 31$

pro třetí kolo: y[j] = 5c4dd124, pro $32 \leq j \leq 47$

Slabina algoritmu byla nalezena celkem brzy po jeho publikování. Merkle ve své nepublikované práci poukazuje na fakt, že je-li přeskočeno třetí kolo u zpracování bloků, lze snadno konstruovat zprávy, jejichž výstup se liší pouze ve 3 bitech [93, s. 192]. Obdobný útok publikovali Bosselaers a Boer. V [21] ukazují, jak je možné konstruovat kolize s vynecháním prvního kola. Na podzim roku 1995 byl publikován útok Hanse Dobbertina [34], ve kterém ukazuje jak nalézt kolizi během pár minut na běžném PC s počtem volání kompresní funkce 2^{20}. Jedná se o dnes již celkem populární útok, ve kterém All Blowfish prodává dům Ann Bonafide. V roce 2005 byl útok ještě vylepšen [125] a bylo dosaženo komplexity 2^8. Stejného roku prezentovali útok také Naito a kol. [82], který docílil komplexity méně než 3 opakování hašovacích operací funkce. V roce 2008 byl publikován teoretický útok [67] proti jednocestnosti funkce, přičemž nalezení zprávy by podle tohoto útoku mělo vyžadovat 2^{102} operací.

Další z rodiny, funkce MD5 byla vytvořena v roce 1991 a publikována, stejně jako obě předchozí funkce, jako RFC 1321 [104]. Jedná se o vylepšenou a o něco pomalejší verzi MD4.

Funkce poskytuje také 128-bitový výstup a předzpracování probíhá stejně jako u MD4. Jako iniciační hodnoty jsou opět definována 4 32-bitová slova (A, B, C, D):

A = 01234567, B = 89abcdef, C = fedcba98, D = 76543210

Na rozdíl od MD4 je přidáno jedno kolo zpracování bloku zprávy, a jsou definovány čtyři pomocné funkce:

$f(X, Y, Z) = (X \wedge Y) \vee (\neg X)\, Z$

$g(X, Y, Z) = (X \wedge Z) \vee Y\, (\neg Z)$

$h(X, Y, Z) = X \oplus Y \oplus Z$

$i(X, Y, Z) = Y \oplus (X \vee (\neg Z))$

Přidaná konstanta:

y[j] = prvních 32 bitů absolutní hodnoty $(\sin (j+1))$ pro $0 \leq j \leq 63$ (j je v radiánech)

Definovaný pořádek pro vstup slov:

z[j] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], pro $0 \leq j \leq 15$

z[j] = [1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12], pro $16 \leq j \leq 31$

z[j] = [5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2], pro $32 \leq j \leq 47$

z[j] = [0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9], pro $48 \leq j \leq 63$

Definovaný počet bitů pro levostrannou rotaci:

s[j] = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22], pro $0 \leq j \leq 15$

s[j] = [5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20], pro $16 \leq j \leq 31$

s[j] = [4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23], pro $32 \leq j \leq 47$

s[j] = [6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21], pro $48 \leq j \leq 63$

Vlastní zpracování zprávy po blocích probíhá následujícím způsobem:

(*initialize working variables*) (A, B, C, D) ← (H_1, H_2, H_3, H_4)

(*Round 1*) For j from 0 to 15 do the following:

t ← (A + f(B, C, D)+ m[z[j]]+y[j]), (A, B, C, D) ← (D, B + t <<<s[j] , B, C)

(*Round 2*) For j from 16 to 31 do the following:

t ← (A + g(B, C, D)+ m[z[j]]+y[j]), (A, B, C, D) ← (D, B + t <<<s[j], B, C)

(*Round 1*) For j from 32 to 47 do the following:

t ← (A + h(B, C, D)+ m[z[j]] + y[j]), (A, B, C, D) ← (D, B + t <<<s[j], B, C)

(*Round 1*) For j from 48 to 63 do the following:

t ← (A + i(B, C, D) + m[z[j]] + y[j]), (A, B, C, D) ← (D, B + t <<<s[j], B, C)

(*update chaining values*) (H_1, H_2, H_3, H_4) ← (H_1+ A, H_2 + B, H_3 + C, H_4 + D)

Zpracování výstupu je opět stejné jako u MD4 (výstupem je zřetězení hodnot H_1 až H_4). Krom přidaného kola zpracování bloku zprávy, a tedy i přidání čtvrté pomocné funkce *i*, můžeme vidět ještě rozdíl v definování funkce *g*. Také přidaná konstanta je, oproti MD4, unikátní pro každý krok zpracování.

Na konferenci Crypto 93 přišli Boer a Bosselaers se způsobem, jak nalézat pseudo-kolize pro tuto funkci, resp. přišli se způsobem jak nalézt 2^{16} kolizí pro první dvě kola kompresní funkce. V roce 2004 předložili Wangová a kol. na konferenci Crypto kolidující zprávy pro MD5. Samotná metoda nalézání kolizí byla publikovaná o rok později [126]. V roce 2005 přišel s metodou nalézání kolizí také Klíma [62]. Klímova metoda, jak sám uvádí, je 3-6x rychlejší než metoda Wangové a je realizovatelná na běžném notebooku. Na jaře roku 2006 Klíma [63] metodu ještě vylepšil a zkrátil čas hledání kolize z osmi hodin na pouhou 1 minutu.

Funkce MD5, ačkoli prolomená, je stále součástí protokolu SSL. Je implementovaná ve standardu RCF 2104 [75], využívaná k tvorbě autentizačních kódů HMAC a celkově má stále mnoho použití.

Posledním přírůstkem v rodině funkcí MD je funkce MD6. Funkce vznikla v roce 2008 a podílela se na ní celá řada autorů v čele s Rivestem. Funkce byla zaslána jako kandidát do soutěže NIST (viz. kapitola 5), kde také postoupila do prvního kola. Rivest však funkci v následujícím roce ze soutěže stáhl, z důvodu její nedostatečné rychlosti. Jak uvedl ve svém příspěvku do oficiální emailové konference soutěže, MD6 potřebuje výrazně zrychlit, aby se minimálně vyrovnala současnému standardu SHA-2. Toto zrychlení vyžaduje snížení současného počtu kol zpracování kompresní funkce, což je 80-168, na 30-40. Při tomto snížení by však hrozilo, že funkce nebude odolná vůči simulovaným útokům, kterým jsou kandidátské funkce podrobovány [110].

Funkce poskytuje výstup od 1 do 512 bitů, je primárně konstruovaná pro 64-bitové platformy avšak, je snadno implementovatelná i na platformy pracující s menším počtem slov. Využívá jednu kompresní funkci, která zpracovává blok zprávy o velikosti 4096 bitů do bloku o velikosti 1024 bitů (bez ohledu na konkrétní požadovanou délku výstupu). Funkce má čtyři vstupy: klíč, počet kol, kontrolní slovo, a unikátní ID. Je založena na stromové struktuře, což umožňuje efektivní paralelní zpracování. Pro dosažení konkrétní délky výstupu je využívána druhá kompresní funkce. Jedná se tedy o formu konstrukce „wide-pipe“.

4.3. Rodina funkcí SHA

SHA je rodina funkcí vytvořená pro účely standardizace bezpečné komunikace a implementace do schématu digitálního podpisu. Rodina funkcí má v současnosti 7 členů SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 a SHA-512/t. Zkratka SHA představuje „Secure Hash Algorithm“ číslo za pomlčkou potom u prvních dvou verzi algoritmu. U ostatních pěti reprezentuje číslo za pomlčkou délku výstupu funkce. Pro zachování sériového označení verze algoritmu jsou tedy funkce SHA-224 až SHA-512 souhrnně neoficiálně označovány jako SHA-2, jelikož prakticky se jedná o tentýž algoritmus. Rozdíl mezi nimi spočívá pouze v různých iniciačních hodnotách, délce bloku a výsledném zobrazení.

Původní funkce SHA byla vytvořena americkou NSA (National Security Agency) a publikována byla NIST jako Federal Information Processing Standard Publication 180 (FIPS

PUB) „Secure Hash Algorithm“ v květnu 1993. Funkce byla velmi brzy upravena a v dubnu 1995 byl publikován nový standard pod označením FIPS PUB 180-1 [38].

Funkce SHA tedy byla nahrazena funkcí SHA-1 a k původní verzi tak přibylo dodatečné sériové označení „0“. Funkce SHA-0 (stejně jako její upravená verze SHA-1) je založena na funkci MD4.

SHA-0 poskytuje výstup o velikosti 160 bitů a zpracovává zprávy o velikosti $\leq 2^{64}$ po blocích o velikosti 512 bitů. Funkce využívá Merkle-Damgårduv iterativní princip. Proces zarovnání je stejný jako u MD4. Bloky zpráv tedy tvoří 16 32-bitových slov (m_0,m_1..m_{15}). Blok zprávy je zpracováván kompresní funkcí ve čtyřech kolech po dvaceti krocích, kde je definováno následující:

Pět 32-bitových slov jako iniciační hodnoty:

H_1= 67452301, H_2 = efcdab89, H_3 = 98badcfe, H_4 = 10325476, H_5 = e3d2e1f0

Přidané konstanty:

pro první kolo: K_1 = 0x56827999, $0 \leq j \leq 19$

pro druhé kolo: K_2 = 0x6ed6dba1, $20 \leq j \leq 39$

pro třetí kolo: K_3 = 0x8fabbcdc, $40 \leq j \leq 59$

pro čtvrté kolo: K_4 = 0xca62c1d6, $60 \leq j \leq 79$

Pomocné operace:

pro první kolo: $F_1(x, y, z,) = (x \wedge y) \vee (\neg x \wedge z)$, $0 \leq j \leq 19$

pro druhé kolo: $F_2(x, y, z,) = x \oplus y \oplus z$, $20 \leq j \leq 39$

pro třetí kolo: $F_3(x, y, z,) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$, $40 \leq j \leq 59$

pro čtvrté kolo: $F_2(x, y, z,) = x \oplus y \oplus z$, $60 \leq j \leq 79$

Slova bloku zprávy jsou nejprve expandována:

$m_i = m_{i-3} \oplus m_{i-8} \oplus m_{i-14} \oplus m_{i-16}$, pro $16 \leq j \leq 79$ $j = i$

a následuje zpracování, které tvoří smyčka:

(*initialize working variables*) (a, b, c, d, e) ← (H_1, H_2, H_3, H_4, H_5)

(*Round 1*) For j from 0 to 19 do the following:

t ← ((a <<< 5) + F_1(b, c, d) + e + K_1 + m_i),

(a, b, c, d, e) ← (t, a, b <<< 30, c, d)

(*Round 2*) For j from 20 to 39 do the following:

t ← ((a <<< 5) + F_2(b, c, d) + e + K_2 + m_i),

(a, b, c, d, e) ← (t, a, b <<< 30, c, d)

(*Round 3*) For j from 40 to 59 do the following:

$t \leftarrow ((a <<< 5) + F_3(b, c, d) + e + K_3 + m_i)$,

$(a, b, c, d, e) \leftarrow (t, a, b <<< 30, c, d)$

(*Round 4*) For j from 60 to 79 do the following:

$t \leftarrow ((a <<< 5) + F_2(b, c, d) + e + K_4 + m_i)$,

$(a, b, c, d, e) \leftarrow (t, a, b <<< 30, c, d)$

(*update chaining values*)

$(H_1, H_2, H_3, H_4, H_5) \leftarrow (H_1 + a, H_2 + b, H_3 + c, H_4 + d, H_5 + e)$.

Výsledné zobrazení je rovno $H_1 \| H_2 \| H_3 \| H_4 \| H_5$ po zpracování všech bloků zprávy *M*.

Funkce SHA-1 se liší od SHA-0 pouze v expanzi zprávy, kdy je přidán cyklický posun o 1 bit vlevo. Expanze tedy pro i = 16 až 79 vypadá: $m_i = ((m_{i-3} \oplus m_{i-8} \oplus m_{i-14} \oplus m_{i-16}) <<< 1)$.

Zároveň je ve standardu FIPS PUB 180-1 uvedena i alternativní metoda výpočtu SHA-1 otisku. Tato alternativní metoda vynechává expanzi zprávy a zavádí konstantu MASK = 0000000f, která je použita pro kroky $j \geq 16$, z níž potřebná slova m získám takto:

$s = j \wedge \text{MASK}$

pro $j \geq 16$ $m_s = ((m_{(s+13)} \wedge \text{MASK} \oplus m_{(s+8)} \wedge \text{MASK} \oplus m_{(s+2)} \wedge \text{MASK} \oplus m_s) <<< 1)$

Oba způsoby výpočtu vedou ke stejnému výsledku

Funkce SHA-0 byla prolomena v roce 2004, kdy Joux neformálně přednesl na Crypto metodu umožňující nalezení kolize s počtem operací 2^{51}. O rok později přišli Wangová a kol. [127] s útokem umožňující nalézání kolizí se složitostí menší než 2^{39} operací.

V roce 2000 představila NIST tři nové funkce s delší délkou výstupu, funkci SHA-256, SHA-384 a SHA-512. Funkce byly publikovány v roce 2002 ve standardu FIPS PUB 180-2 [39]. Motivací zavedení těchto funkcí bylo poskytnutí stejné úrovně bezpečnosti proti nalezení kolizí, jaká byla očekávána od tří standardizovaných délek klíčů (128, 192, 256 bitů) nově vybrané šifry Rijndael publikované jako AES (Advanced Encryption Standard), který měl nahradit DES, jenž byl v roce 1999 prolomen.

V roce 2004 vydal NIST upozornění o změně standardu, ve kterém implementuje do FISP PUB 180–2 ještě funkci SHA-224. Důvodem doplnění bylo vytvořit kompletní standardizovanou sérii hašovacích funkcí tak, aby naplňovala definované úrovně zabezpečení 80, 112, 128, 192 a 256 bitů. Nový oficiální standard, který již obsahoval funkci SHA-224 byl vydán až v říjnu 2008 [40].

Funkce SHA-224 a SHA-256 zpracovává zprávy o velikosti $0 \leq l < 2^{64}$ bitů po blocích o velikosti 512 bitů. Zarovnání je prováděno stejně jako u SHA-1. Tedy je připojen jeden 1bit a

poté co nejmenší počet nulových bitů, tak aby platilo $l + 1 + k = 448 \bmod 512$. Posledních 64 bitů je opět vyjádřením délky zprávy l.

Funkce SHA-384 a SHA-512 zpracovává zprávy o velikosti $0 \leq l < 2^{128}$ bitů po blocích o velikosti 1024 bitů. Zarovnání tedy odpovídá $l + 1 + k = 896 \bmod 1024$, jelikož funkce používají k vyjádření délky zprávy dvojnásobný počet bitů (128).

Funkce SHA-224 a SHA-256 pracují se slovy o velikosti 32 bitů a funkce SHA-384 a SHA-512 se slovy o velikosti 64 bitů. Všechny funkce používají, na rozdíl od SHA-1, osm iniciačních hodnot, resp. osm 32-bitových nebo 64-bitových slov. Tyto konstanty jsou pevně definovány jako:

SHA-224

H_1 = c1059ed8
H_2 = 367cd507
H_3 = 3070dd17
H_4 = f70e5939
H_5 = ffc00b31
H_6 = 68581511
H_7 = 64f98fa7
H_8 = befa4fa4

SHA-256

H_1 = 6a09e667
H_2 = bb67ae85
H_3 = 3c6ef372
H_4 = a54ff53a
H_5 = 510e527f
H_6 = 9b05688c
H_7 = 1f83d9ab
H_8 = 5be0cd19

SHA-384

H_1 = cbbb9d5dc1059ed8
H_2 = 629a292a367cd507
H_3 = 9159015a3070dd17
H_4 = 152fecd8f70e5939
H_5 = 67332667ffc00b31
H_6 = 8eb44a8768581511
H_7 = db0c2e0d64f98fa7
H_8 = 47b5481dbefa4fa4

SHA-512

H_1 = 6a09e667f3bcc908
H_2 = bb67ae8584caa73b
H_3 = 3c6ef372fe94f82b
H_4 = a54ff53a5f1d36f1
H_5 = 510e527fade682d1
H_6 = 9b05688c2b3e6c1f
H_7 = 1f83d9abfb41bd6b
H_8 = 5be0cd19137e2179

Bloky zprávy jsou u SHA-224 s SHA-256 zpracovávány v 64 krocích, u SHA-384 a SHA-512 v 80 krocích. V každém kroku je, na rozdíl od SHA-1, přidaná konstanta K, definovaná zvlášť pro každý krok. Konstantu K tvoří pro SHA-224, SHA-256 prvních 32 bitů a pro SHA-384, SHA-512 prvních 64 bitů, frakční části třetích odmocnin prvních šedesátičtyř (pro SHA-

224, 256) nebo osmdesáti (pro SHA-384, 512) prvočísel. Je tedy definováno 64 a 80 hodnot konstanty *K*.

Pro všechny čtyři funkce jsou definovány pomocné funkce:

$F_1(x, y, z,) = (x \wedge y) \oplus (\neg x \wedge z)$

$F_2(x, y, z,) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$

dále pro **SHA-224** a **SHA-256**:

$\sum_0(x) = (2 >>> x) \oplus (13 >>> x) \oplus (22 >>> x)$

$\sum_1(x) = (6 >>> x) \oplus (11 >>> x) \oplus (25 >>> x)$

$\sigma_0(x) = (7 >>> x) \oplus (18 >>> x) \oplus (3 >> x)$

$\sigma_1(x) = (17 >>> x) \oplus (19 >>> x) \oplus (10 >> x)$

pro **SHA-384 a SHA-512** :

$\sum_0(x) = (28 >>> x) \oplus (34 >>> x) \oplus (39 >>> x)$

$\sum_1(x) = (14 >>> x) \oplus (18 >>> x) \oplus (41 >>> x)$

$\sigma_0(x) = (1 >>> x) \oplus (8 >>> x) \oplus (7 >> x)$

$\sigma_1(x) = (19 >>> x) \oplus (61 >>> x) \oplus (6 >> x)$

Zpracování pro **SHA-224** a **SHA-256** probíhá následovně:

zpráva je rozdělena do bloků kde každý blok tvoří 16 32bitových slov $m_{0,}$ $m_{1...}$ m_{15}

blok zprávy je pro kroky *j* 16 až 63 expandován:

pro $16 \leq j \leq 63$ $j = i$

$m_i = \sigma_1(m_{i-2}) + m_{i-7} + \sigma_0(m_{i-15}) + m_{i-16}$

vlastní zpracování tvoří smyčka:

(*initialize working variables*) $(a, b, c, d, e, f, g, h) \leftarrow (H_1, H_2, H_3, H_{4,} H_5, H_6, H_7, H_8)$

For j from 0 to 63 do the following:

$t \leftarrow ((h + \sum_1(e) + F_1(e, f, g) + K_j + m_i) + (\sum_0(a) + F_2(a, b, c,)))$,

$(a, b, c, d, e, f, g, h) \leftarrow (t, a, b, c, d + (h + \sum_1(e) + F_1(e, f, g) + K_j + m_i), e, f, g)$

(*update chaining values*)

$(H_1, H_2, H_3, H_{4,} H_5, H_6, H_7, H_8) \leftarrow (H_1 + a, H_2 + b, H_3 + c, H_4 + d, H_5 + e, H_6 + f, H_7 + g,$

$H_8 + h)$.

Pro SHA-256 tvoří výsledné zobrazení $H_1 \| H_2 \| H_3 \| H_4 \| H_5 \| H_6 \| H_7 \| H_8$ po zpracování všech bloků zprávy *M*. U SHA-224 je vynecháno poslední 32 bitů a zobrazení tedy tvoří pouze $H_1 \| H_2 \| H_3 \| H_4 \| H_5 \| H_6 \| H_{7.}$

Algoritmus zpracování u SHA-384 a SHA-512 je stejný. Pouze počet kroků *j* není 64, ale 80. Výsledné zobrazení pro SHA-512 je opět tvořeno zřetězením hodnot H_1 až H_8 po zpracování všech bloků. Pro SHA-384 je (obdobně jako u SHA-224) oříznuto posledních 128 bitů a zobrazení tvoří pouze zřetězení H_1 až H_6.

V roce 2005 obdržela rodina bezpečných hašovacích algoritmů silnou ránu. Kdy na konferenci Crypto zveřejnili Wangová a kol. [128] metodu nalézání kolizí pro SHA-1 se složitostí 2^{69}. Tento útok Wangová v zápětí vylepšila a dosáhla složitosti 2^{63}. K celé situaci vydal NIST na svých stránkách komentář [85], ve kterém říká, že ačkoli nejsou známy prakticky potvrzené výsledky této metody, považuje útok se složitostí nalezení kolize 2^{63} za proveditelný, a tudíž uznává, že Wangová nalezla praktickou metodu nalézání kolizí pro SHA-1. Dále vyhlašuje konání workshopu, na kterém budou zhodnoceny důsledky útoku Wangové, a oznamuje, že vládní agentury musejí přestat používat digitální podpisy založené na SHA-1 do konce roku 2010.

Workshopy byly konány dva (konec roku 2005 a polovina roku 2006) a na jejich základě NIST vyhlásil veřejnou soutěž o nový hašovací algoritmus, který bude zařazen mezi bezpečné hašovací standardy (blíže kapitola 5).

Zajímavé je, že v březnu 2012 vydal NIST novou verzi standardu „FIPS PUB 180-4“ [41]. Tedy ještě před oficiálním vyhlášením výsledků soutěže, přičemž nový standard byl zamýšlen právě na základě těchto výsledků a měl být vydán v posledním kvartálu roku 2012.

V této nové verzi stále zůstává funkce SHA-1 a je přidána funkce SHA-512/t, resp. funkce SHA-512/224 a SHA-512/256. Malé písmeno *t* za lomítkem značí počet bitů, o který bude zkrácen výstup funkce SHA-512 a zároveň slouží jako parametr pro generování iniciačních hodnot. Iniciační hodnoty funkce SHA-512/t jsou vypočítávány na základě exkluzivní disjunkce iniciačních hodnot SHA-512 a vyjádření čísla *t*, prostřednictvím SHA-512(„SHA-512/t“).

Konkrétně tedy pro SHA-512/224 a SHA-512/256 to jsou:

H1 = 8C3D37C819544DA2	H1 = 22312194FC2BF72C
H2 = 73E1996689DCD4D6	H2 = 9F555FA3C84C64C2
H3 = 1DFAB7AE32FF9C82	H3 = 2393B86B6F53B151
H4 = 679DD514582F9FCF	H4 = 963877195940EABD
H5 = 0F6D2B697BD44DA8	H5 = 96283EE2A88EFFE3
H6 = 77E36F7304C48942	H6 = BE5E1E2553863992
H7 = 3F9D85A86A1D36C8	H7 = 2B0199FC2C85B8AA

H8 =1112E6AD91D692A1 H8 = 0EB72DDC81C52CA2

Obecným důvodem pro zkracování výstupu hašovacích funkcí je jejich použití. Různé aplikace mohou vyžadovat nestandardní délku hašovacího kódu. V praxi je tedy kód standardní funkce zkracován (ořezán) odebráním určitého počtu bitů tak, aby bylo dosaženo délky potřebné pro aplikaci. Je-li toto zkrácení provedeno nestandardní metodou, je tím snížena i bezpečnost hašovacího kódu, kterou garantuje standardní použití hašovací funkce.

Proč je tedy definována funkce SHA-512/256, když stejně tak mohu použít funkci SHA-256, přičemž obě funkce poskytují výstup 256 bitů a tedy bezpečnost 128 bitů.

Důvodem definování těchto verzí SHA-512, jak uvádí Schneier [119], byl fakt, že funkce SHA-512 pracuje na 64-bitových procesorech rychleji než SHA-256.

4.4. Shrnutí

Jak uvádí Preneel [97], do roku 1993 bylo známo zhruba 40-60 hašovacích funkcí a dalších 30-40 vzniklo do roku 2008. Z tohoto počtu lze odvodit, že funkce uvedené v této kapitole, rozhodně nemohou být všechny funkce, které vznikly v období mezi lety 1989 až 2008.

Hašovací funkce uváděné v této kapitole, patří mezi funkce, které jsou určitým způsobem význačné. Výše uvedené funkce jsou buď zakotvené v určitém standardu (viz. funkce SHA, MDx, RIPEMD, Whirlpool, GOST aj.), nebo odrážejí určitý specifický přístup pro tvorbu hašovacích funkcí, případně jsou významné jiným způsobem.

Samotné období bylo definováno ve vztahu ke krizovému roku 2005. V tomto roce byly oficiálně prolomeny dvě celosvětově nejrozšířenější a nejpoužívanější hašovací funkce (MD5 a SHA-1), což vedlo k potřebě přehodnotit stav standardů v této oblasti, a vyvodit důsledky, které z této skutečnosti vyvstávají. Jedním z těchto důsledků bylo vypsání veřejné soutěže o nový hašovací standard (2007), což do jisté míry přispělo k urychlení vývoje nových hašovacích funkcí.

5. Nový hašovací standard

Na podzim roku 2007 vyhlásil NIST veřejnou soutěž o vytvoření nového hašovacího algoritmu, který by rozšířil stávající rodinu hašovacích standardů. Důvodem potřeby nového hašovacího algoritmu, jak NIST uvádí, jsou vážné útoky proti SHA-1, které byly zaznamenány v posledních letech a také fakt, že některé dosud užívané hašovací funkce byly prolomeny.

Skutečnost, že soutěž je koncipována, jako veřejná není na poli kryptografie nic nového, stejný způsob použil NIST již v roce 1997. Tehdy bylo snahou nalézt nový standard pro globální informační bezpečnost. Vítězem této soutěže se stala v roce 2001 symetrická bloková šifra Rijndael. Obdobného charakteru byl i okrajově zmíněný evropský projekt NESSIE.

Celá myšlenka soutěž vznikla na základě dvou pořádaných workshopů. První workshop byl pořádán na přelomu října a listopadu 2005. Na tomto workshopu byl řešen zejména dopad útoku proti SHA-1. Respektive skutečnost, do jaké míry ovlivní tento útok aplikace, které tuto funkci používají, za jakých podmínek bude nutné doporučit okamžité zrušení používání této funkce, a zda bude skutečně prakticky možný přechod digitálních podpisů na některou z funkcí SHA-2 do konce roku 2010. Zejména v posledním bodě, tedy v souvislosti s digitálním podpisem, bylo namítáno, že v současných softwarových aplikacích zatím není dostatečná podpora funkcí SHA-2. Tudíž podpisy na základě SHA-1 nebo MD5 sice mohou být novými aplikacemi odmítnuty, avšak lidé je budou nadále používat, jelikož staré aplikace většinou neumožňují generovat nová data prostřednictvím funkcí SHA-2 [88].

V srpnu následujícího roku byl konán druhý workshop. Zde byla probírána zejména nová struktura hašovacích funkcí a hašovací funkce v praxi. Diskutovány byly i vlastnosti hašovacích funkcí a jaké vlastnosti by hašovací funkce měla mít, aby byla odolná v dlouhodobém časovém horizontu. Dalším faktorem bylo hodnocení, zda je lepší vytvářet jednu obecně-účelovou hašovací funkci nebo několik funkcí specifických.

Implementace jedné hašovací funkce představuje rozhodně menší nároky na čas a finance z pohledu výrobců softwaru, institucí a obecně zainteresovaných osob, nicméně při jejím prolomení se bude opakovat stejná situace jako u SHA-1.

Krom těchto bodů bylo zvažováno přijetí některých již existujících funkcí, konkrétně RadioGatún, LASH a Endor-R, jako náhradu za prolomené funkce.

Na základě tohoto workshopu, bylo učiněno rozhodnutí o vytvoření nového standardu, který bude znám pod názvem SHA-3.

Přepokládaný časový harmonogram pro realizaci byl následující [86]:

říjen – prosinec 2006:	Vypracování předběžných minimálních požadavků na kandidátské funkce, požadavků na dokumentaci a předběžné stanovení evaluačních kritérií.
leden – březen 2007:	Publikování vypracovaného materiálu, jeho prezentace na konferenci RSA a FSE 2007 a otevření veřejné diskuse.
duben – září 2007:	K datu 27. dubna 2007 ukončit veřejnou diskuzi a vyhodnotit podněty.
říjen – prosinec 2007:	Finalizace a publikování evaluačních kritérií, minimálních požadavků na funkce a na dokumentaci. Veřejné vyhlášení soutěže.
říjen – prosinec 2008:	Uzávěrka přihlášek do soutěže.
duben – červen 2009:	Zhodnocení přijatých kandidátských funkcí a vybrání kandidátů, kteří splňují minimální požadavky, publikované během října - prosince 2007. Uspořádání první kandidátské konference, na které budou oznámeny funkce, které byly vybrány pro postup do prvního kola soutěže a zároveň otevření veřejné diskuse nad funkcemi vybranými pro první kolo.
duben – červen 2010:	Ukončení veřejné diskuse. Uspořádání druhé kandidátské konference, kde budou zhodnoceny výsledky analýz kandidátských funkcí.
červenec – září 2010:	Adresování veřejných komentářů tvůrcům kandidátských funkcí. Vybrání finalistů. Příprava reportu, ve kterém bude vysvětlen výběr finalistů. Veřejné oznámení finalistů, publikování reportu a otevření veřejné diskuze nad finalisty.
říjen – prosinec 2010:	Přijetí případného vylepšení finálních kandidátských funkcí a začátek finálního kola soutěže.
říjen – prosinec 2011:	Ukončení veřejné diskuse nad finalisty.
leden – březen 2012:	Uspořádání finální konference, kde budou diskutovány funkce vybrané pro finále, včetně komentářů vzešlých z veřejné diskuse.
duben – červen 2012:	Volba vítěze a příprava reportu, popisujícího finální volbu. Oznámení nové (nových) hašovací/ch funkce (funkcí) vybrané (vybraných) pro účely standardu.

červenec – září 2012:	Návrh revize hašovacího standardu a publikování tohoto návrhu pro veřejné komentáře.
říjen – prosinec 2012:	Ukončení veřejné diskuse nad návrhem a zaslání navrhovaného standardu na ministerstvo obchodu k podpisu.

Základní požadavky na algoritmus byly definovány z hlediska dostupnosti, implementace a velikosti výstupu a byly publikovány v listopadu jako FRN-Nov07 [84].

Algoritmus musí být veřejně publikovatelný a dostupný celosvětově bez poplatků, jeho implementace musí být z hlediska hardwaru a softwaru proveditelná v širokém rozsahu, musí podporovat výstup o velikosti 224, 256, 384 a 512 bitů přičemž maximální délka zprávy musí být nejméně 264 bitů.

Kritéria pro hodnocení byla stanovena jako bezpečnost, náročnost a schopnost implementace.

Z pohledu bezpečnosti mají být algoritmy posuzovány na základě jejich odolnosti vůči kolizím, na základě skutečné bezpečnosti, kterou poskytují, ve srovnání s jinými algoritmy poskytujícími stejnou délku výstupu. Pro účely bezpečnosti je také hodnocena míra schopnosti algoritmu chovat se jako náhodné orákulum a matematické základy, na nichž je algoritmus postaven. Také bylo doporučeno návrhářům, aby funkce využívali jinou než Merkle-Damgårdovu konstrukci. Důvodem byly generické útoky (Joux [58], Kelsey a Schneier [60]), které obecně snižují bezpečnost funkcí, které tuto konstrukci používají.

Náročností se rozumí zejména paměťová náročnost, tedy požadavky na paměť ve smyslu hardwarové a softwarové implementace, což úzce souvisí s posledním kritériem, tj. schopností implementace.

V tomto bodě se posuzuje zejména flexibilita algoritmu, tedy jednak schopnost algoritmu efektivně a bezpečně pracovat na širokém spektru platforem, ale třeba také schopnost paralelního zpracování dat pro dosažení vyšší efektivity.

5.1. První a druhé kolo soutěže

Do soutěže bylo k 1. listopadu 2008 přihlášeno celkem 64 algoritmů hašovacích funkcí. Z těchto bylo na počátku prosince 2008 vybráno celkem 51 jako postupujících do prvního kola. Jak uvádí Preneel [97], není překvapující, že ze třinácti zamítnutých bylo 5 velmi rychle prolomeno. Z postupujících jednapadesáti bylo devět prolomeno ještě před konáním první oficiální konference kandidátů.

Během soutěže NIST používal k hodnocení kandidátů hodnotící kriteria specifikovaná v [84]. Tato kriteria byla diskutována a dále vyjasněna na první konferenci kandidátů konané v Katholieke Universiteit, Leuven v Belgii únoru 2009, kde byly představeny všechny

kandidátské funkce. Za relativní pořadí důležitosti kriterií NIST při výběru kandidátů považoval bezpečnost, náklady, algoritmus a implementační charakteristiky v daném podání:

Bezpečnost

⇒ použití hašovacích funkcí (algoritmů)

⇒ specifické požadavky na použití hašovacích funkcí (algoritmů) pro tvorbu autetizačních kódů (HMAC), pseudo-náhodné funkce (PRF) nebo náhodného hašování (Randomized Hashing)

⇒ přidaná bezpečnost hašovací funkce

⇒ vyhodnocení týkající se odolnosti proti útokům a

⇒ ostatní faktory, které lze vzít v úvahu

Náklady a výkonnost

⇒ výpočetní efektivitu, která se vztahuje k rychlosti algoritmu a

⇒ nároky na paměť, které zahrnují velikost kódu, požadavky na RAM pro softwarovou implementaci, stejně jako počet bran (zápisů na bráně) pro hardwarovou implementaci.

Algoritmus a implementační charakteristiky

⇒ flexibilita a jednoduchost designu

Pro vyhodnocení bezpečnosti NIST (krom analýz vlastních kryptoanalytiků) prostudoval velké množství zpětných reakcí získaných z řad kryptografické komunity v oficiální e-mailové diskuzi „hash-forum@nist.gov“, stejně jako bezpečnostní argumenty prezentované samotnými návrháři.

V hodnocení softwarové a hardwarové implementace uvažoval NIST různá „benchmarková“ porovnání. Hlavním přínoscem byl projekt ECRYPT „Benchmarking of All Submitted Hashes“ (eBASH), který prováděl měření rychlosti jednotlivých kandidátů v široké paletě 32-bitových a 64-bitových platforem.

eBASH překompiloval návrháři předložené implementace a ostatní části s různými kompilačními volbami, vykazujícími nejlepší výsledky v mediánech a kvartilách pro zprávy v rozsahu od 8 do 4096 bytů a provedl extrapolaci i pro delší zprávy. Byly použity i jiné studie pro měření softwarové výkonnosti. Na příklad benchmarkový projekt (XBX) zhodnotil výsledky měření výkonu na malých zařízeních.

Poslední z hodnotících kriterií se vztahuje k flexibilitě (pružnosti, přizpůsobivosti) a jednoduchosti návrhu. Bylo stanoveno, že návrh s vyšší flexibilitou dostane vždy přednost a zároveň, že uchazeči budou posuzováni podle relativní jednoduchosti návrhu.

Záměrem bylo povzbudit snadnou pochopitelnost a analyzovatelnost návrhu a dosáhnout tak větší důvěry ve jeho vlastní bezpečnost.

Vyhlášení kandidátů postupujících do druhého kola, proběhlo v červenci 2009. Celkem do druhého kola postoupilo 14 kandidátů. Těmito kandidáty byli BLAKE, Blue Midnight Wish, CubeHash, ECHO, Fugue, Grøstl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD a Skein. Tento výběr do dalšího kola byl dán mimo jiné i snahou o pestrost a různost návrhů. Snaha NIST byla vybrat takové kandidáty, jejichž vnitřní konstrukce bude různorodá a, tudíž aby pro techniku jednoho útoku bylo velmi nepravděpodobné, že prolomí sadu návrhů všech finalistů.

Stručně tedy jednotlivé funkce:

⇒ Funkce **BLAKE** [1] je založena na konstrukci HAIFA a je koncipována jako rodina hašovacích funkcí. Funkce zpracovává zprávy o velikosti 2^{64} a 2^{128} bitů. Funkce BLAKE-32, BLAKE-64, BLAKE-28, BLAKE-48 se od sebe liší různými iniciačními hodnotami a různými přidanými konstantami *c*. BLAKE-32 používá stejné iniciační hodnoty jako SHA-256 a 16 konstant, které jsou tvořeny prvními čísly π. Funkce zpracovává slova o velikosti 32 bitů a poskytuje výstup 256 bitů. Jeden blok zprávy tvoří 16 slov. Kompresní funkce zpracovává bloky v deseti kolech, přičemž využívá pro jednotlivá kola fixně stanovenou permutaci σ. Vstup kompresní funkce tvoří řetězící hodnota *h*, blok zprávy *m*, sůl *s* a counter *t*. Sůl tvoří libovolný 128-bitový řetězec, což umožňuje použití BLAKE pro tvorbu MAC kódů. V případě, že sůl není v rámci aplikace potřeba, je hodnota tohoto řetězce stanovena jako 0. Counter představuje 64-bitový řetězec, jenž vyjadřuje počet již zpracovaných bitů zprávy. Celkově vstup tvoří 30 slov (tj. 960 bitů – 16 slov blok zprávy, 4 slova sůl, 2 slova counter, 8 slov řetězící hodnota). Každé kolo tvoří celkem 8 operací před změnou vnitřního stavu, který představuje matice 4 x 4 slov v_0-v_{15}. Prvních 8 slov tvoří řetězící hodnoty *h*, další čtyři slova vzniknou na základě XOR operace mezi solí a prvními čtyřmi konstantami (c_0-c_3), poslední čtyři slova vzniknou na základě XOR operace mezi countrem a dalšími čtyřmi konstantami (c_4-c_7).

Operace jsou rozděleny do dvou setů pro jednotlivá *G*, která přestavují pořadí slov vnitřního stavu, přičemž každé G je tvořeno čtyřmi slovy. Pro G_0-G_3(a, b, c, d) je to:

$a \leftarrow a + b + (m_{\sigma r(2i)} \oplus c_{\sigma r(2i+1)})$, $b \leftarrow (b \oplus c) >>> 12$, $c \leftarrow c + d$, $d \leftarrow (d \oplus a) >>> 16$.

Pro G_4-G_7(a, b, c, d):

$a \leftarrow a + b + (m_{\sigma r(2i+1)} \oplus c_{\sigma r(2i)})$, $b \leftarrow (b \oplus c) >>> 7$, $c \leftarrow c + d$, $d \leftarrow (d \oplus a) >>> 8$.

Celkový výstup vznikne zřetězením osmi hodnot, které jsou vždy produktem XOR operace mezi řetězící hodnotou *h*, solí a dvěma slovy vnitřního stavu *v*.

BLAKE-64 pracuje stejně. Rozdílem jsou iniciační hodnoty (stejné jako SHA-512), konstanty, počet kol zpracování (14), dvojnásobná délka proměnných (délka řetězící hodnoty 512 bitů, blok 1024 bitů, sůl 256 bitů, counter 128 bitů) a jinak definované operace (resp. posuny v rámci operací). Celkově poskytuje BLAKE-64 výstup 512 bitů.
Funkce BLAKE-24 a BLAKE-48 užívají stejné iniciační hodnoty jako SHA-224 a SHA-348. Pracují stejně jako BLAKE-32 a BLAKE-64, přičemž jejich výstup představuje pro BLAKE-24 výstup = BLAKE-32 – 32bitů a stejně BLAKE-48 = BLAKE-64 – 128 bitů.

⇒ Funkce **Blue Midnight Wish** (BMW) [49] je založena na Merkle-Damgårdově konstrukci ve formě wide-pipe. Jedním ze spolutvůrců této funkce je i český kryptolog Vlastimil Klíma. Trochu zvláštní název byl pravděpodobně volen, kvůli akronymu BMW, který vede k určité analogii a naznačuje, že funkce pracuje velmi rychle. BMW používá tři kompresní funkce. Kompresní funkci tvoří tři subfunkce. Je přidáno jedno zvláštní kolo finálního zpracování. Funkce je koncipovaná jako rodina hašovacích funkcí. Zpracovává zprávy o velikosti 2^{64} bitů. Velikost bloků a slov je pro jednotlivé verze shodná s SHA-2. Funkce používá dvě expanze zprávy $expand_1$ a $expand_2$, různé fixní iniciační hodnoty pro každou verzi a 13 posuvných operací. Zarovnání je pro verze 224 a 256 bitů shodné jako u SHA-2. U verze 384 a 512 bitů je pro délku zprávy rezervováno 64, tedy stejný počet jako u verze 224 a 256 (zde SHA-2 používá 128 bitů).

⇒ Funkce **CubeHash** [10] je koncipována jako jedna hašovací funkce a je částečně založena na sponge konstrukci. Jejím tvůrcem je Daniel J. Bernstein. Používá celkem pět parametrů, počet iniciačních kol, počet kol zpracování na jeden blok zprávy, počet bytů na jeden blok, počet kol pro finalizaci a délka výstupu. Funkce zpracovává zprávu o velikosti 0 až 2^{128}-1 bitů. Délka výstupu funkce je volitelná dle daného parametru. Pracuje s bloky (stavy) o velikosti 128 bytů a slovy o velikosti 4 byty (tedy 32 4-bytových slov). Zpracování stavu je dáno fixní permutací *T*.

⇒ Funkce **ECHO** [8] je striktně založená na AES a využívá konstrukci HAIFA. Pracuje se slovy o velikosti 128 bitů, která jsou prezentována jako matice 4 x 4 slov. V závislosti na délce výstupu používá buď 512-bitovou (pro výstup 224 a 256 bitů), nebo 1024-bitovou (pro výstup 384 a 512 bitů) kompresní funkci. Velikost bloku zprávy je buď 1536, nebo 1024 bitů. Zpracování je dáno jako $V_i = Compres_x (V_{i-1}, M_i, C_i, SALT)$.
Sůl tvoří řetězec o velikosti 128 bitů a C_i (counter) poskytuje řetězec o délce buď 64, nebo 128 bitů.

⇒ Funkce **Fugue** [50] byla vytvořena týmem převážně z IBM. Je koncipována jako jedna hašovací funkce. Funkce byla inspirována algoritmem Grindhal. Je založená na Sponge konstrukci. Pracuje s pěti parametry *F[n, s, k, r, t]*. Parametry jsou, počet slov ve výstupu, počet sloupců vnitřního stavu, počet sub-kol na jedno kolo transformace, počet kol první fáze finalizace a počet kol druhé fáze finální transformace.

⇒ Funkce **Grøstl** [46] je koncipována jako jedna hašovací funce. Používá wide-pipe Merkle-Damgårdovu konstrukci a zvláštní transformaci na výstupu. Tvůrcem je tým autorů, mezi nimiž jsou i Gauravaram a Kundsen. Kompresní funkci tvoří dvě fixní permutace *P* a *Q*, které jsou podobné jako u AES. Funkce zpracovává zprávu prostřednictvím osmi kroků a poskytuje výstup od 8 do 512 bitů. Výsledné zobrazení tvoří zkrácení výstupu kompresní funkce o *n* bitů. Samotná funkce je definována jako: $f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h$.

⇒ Funkce **Hamsi** [66] je koncipována jako jedna hašovací funkce. Používá techniku zřetězeného permutovaného krácení podobně jako Grindhal. Standardními výstupy jsou 256 a 512 bitů.

⇒ Funkce **JH** [130] je koncipována jako rodina hašovacích funkcí. Je částečně založena na wide-pipe Merkle-Damgård. Jednotlivé verze jsou rozlišeny pouze volbou iniciačních hodnot. Používá jednu kompresní funkci F_8, která je konstruovaná jako bloková šifra a využívá fixní permutaci E_8. E_8 tvoří 7 různých P-permutací, jedna 8-bitová L-permutace a 4-bitové S-boxy.

⇒ Funkce **Keccak** [13] je založen na konstrukci Sponge, přičemž autoři (Bertoni, Daemen, Peeters, van Assche) této hašovací funkce jsou i autory právě této konstrukce. Funkce Keccak je následníkem RadioGatún, která byla vytvořena touto (mimo Peeterse) trojicí v roce 2006.
Základní verze Keccak operuje s vnitřním stavem o velikosti 1600 bitů. Používá operace XOR, AND, NOT a rotační posuny. Pro verzi nového standardu jsou doporučeny hodnoty:
pro verzi 224 bitů r = 1152 bitů, c = 448 bitů a d = 28,
pro verzi 256 bitů r = 1088 bitů, c = 512 bitů a d = 32,
pro verzi 384 bitů r = 832 bitů, c = 768 bitů a d = 48,
pro verzi 512 bitů r = 576 bitů, c = 1024 bitů a d = 64.
Funkce využívá definovaných 7 permutací. Princip zpracování je totožný s výše popsanou konstrukcí Sponge. Vnitřní stav je zpracováván jako třídimenzionální oblast, která je dána jako *s[w(5y + x) + z] = a[x][y][z]*. Proměnné *x, y, z*, představují bitové pozice indexované od 0. Každé kolo zpracování tvoří 5 kroků.

⇒ Funkce **Luffa** [26] je založena na Sponge, ve speciální formě wide-pipe. Je koncipována jako jedna funkce. Zpracovává zprávy o velikosti $\leq 2^{64}$ nebo $\leq 2^{128}$ bitů. Blok zprávy je tvořen osmi slovy o velikosti 32 bitů. Pracuje s 256-bitovou nelineární permutací. Tato permutace je rozdělena do *w* paralelních sub-permutací, které jsou aplikovány uvnitř konstrukce. Počet sub-permutací je závislý na délce výstupu. Pro verzi 224 a 256 bitů jsou použity 3 sub-permutace, pro 384 bitů 4 a 512 bitů 5 sub-permutací. Zpracování bloku tedy probíhá v 3-5 kolech.

⇒ Funkce **Shabal** [25] používá určitou specifickou konstrukci, kterou lze považovat variantu wide-pipe Merkle-Damgård. Jsou standardně definované funkce pro výstup 192, 224, 256, 384 a 512 bitů. Používá klíčovanou permutaci. Na rozdíl od ostatních funkcí nepracuje přímo se slovy (jedno slovo 32 bitů). Vnitřní stav tvoří „bloky" *A*, *B*, *C*, přičemž blok *A* tvoří 384 bitů (12 slov), blok *B* 512 bitů (16 slov) a blok *C* též 512 bitů. Iniciační hodnota bloků je různá pro každou verzi. Zpracování zprávy je definováno jako: $(A, B) \leftarrow P_{M,C}(A \oplus W, B + M)$, $(A, B, C) \leftarrow (A, C - M, B)$. *M* představuje blok zprávy a *W* je counter. Po zpracování všech bloků zprávy jsou přidána 3 finální zpracovací kola: $(A, B) \leftarrow P_{M,C}(A \oplus k, B + M_k)$, $(A, B, C) \leftarrow (A, C - M_k, B)$. Zde *k* představuje počet bloků zprávy a M_k je bloku v posledním kroku. M_k a *k* jsou pro poslední tři kola fixními hodnotami.
Funkce používá dva měnitelné parametry *p* a *r*, *p* je počet smyček prováděných v klíčové permutaci a *r* je velikost *A*. Defaultní hodnoty jsou 3 a 12 (*p*, *r*). Přidání na těchto parametrech by mělo odpovídat ekvivalenci $16p \equiv 0 \bmod r$.

⇒ Funkce **SHAvite-3** [20] je založena na konstrukci HAIFA. Kompresní funkce je založena na blokové šifře a Davies-Meyerově konstrukci. Počet kol zpracování bloku je volitelným parametrem. Jako základní je definováno 12 kol pro verze 224 a 256 bitů a 14 kol pro verze 384 a 512 bitů.

⇒ Funkce **SIMD** [68] je částečně založena na wide-pipe Merkle-Damgårdově konstrukci a Davies-Meyerově kompresní funkci. SIMD pracuje velmi podobně jako funkce MDx a SHA. Jsou definovány dvě základní verze pro 256-bitový a 512-bitový výstup. Verze s nižším výstupem jsou získány odebráním definovaného počtu bitů z verzí základních (stejně jako u SHA-2). Iniciační hodnoty jsou různé pro každou verzi. Vnitřní stav je reprezentován maticemi 4 x 4 (pro 256 bitů: *A0-A3*, *B0-B3*, *C0-C3*, *D0-D3*) a 4 x 8 (pro 512 bitů: *A0-A7*, *B0-B7*, *C0-C7*, *D0-D7*) slov, přičemž jedno slovo tvoří 32 bitů. Používá 4 permutace. Blok zprávy tvoří 512 a 1024 bitů (16 a 32 slov) a je použita osminásobná expanze. Každé kolo zpracování tvoří 16 kroků. Výsledný výstup je tvořen slovy vnitřního stavu *A0-B3* (256 bitů), *A0-B2* (224 bitů), *A0-B7* (512 bitů) a *A0-B3* (384 bitů).

⇒ Funkce **Skein** [117] je postavena na blokové šifře Threefish. Používá kompresní funkci založenou konstrukci Matyas-Meyer-Oseas. Funkce poskytuje výstup od 128 do 1024 bitů, přičemž jsou definovány 3 základní verze Skein-256 (poskytuje výstup 128, 160, 224, 256 bitů), Skein-512 (výstup 128, 160, 224, 256, 384, 512 bitů) a Skein-1024 (výstup 384, 512 a 1024 bitů). Jako primární návrh je verze Skein-512. Funkce zpracovává bloky v kolech, jejichž počet je dán velikostí bloku. Počet kol je standardně definován jako 72 pro bloky 256 a 512 bitů a 80 pro 1024 bitů.

5.2. Finále soutěže

V srpnu 2010 proběhla v Santa Barbaře druhá konference kandidátů, kde byly prezentovány kandidátské funkce a jejich případná vylepšení, která byla učiněna pro druhé kolo.

NIST bral v úvahu pozitivní bezpečnostní argumenty a povolil návrhářům jejich algoritmy pro druhé kolo vylepšit (tzn. udělat drobné modifikace). Tato skutečnost byla dána myšlenkou, zda nejlepší z útoků na jednotlivé kandidáty, nelze zmírnit jednoduchou úpravou algoritmu. NIST se obecně spokojil s vylepšeními v počtu cyklů nebo konstant. Více podezřelé by byly úpravy s vlivem na strukturu kompresních funkcí, protože takové úpravy již byly kryptoanalýzami v prvním kole překonány.

Jak se ukázalo, kryptoanalýza blokových šifer je obecně pokročilejší než kryptoanalýza hašovacích algoritmů. Funkce, jejichž bezpečnost je spojena s blokovou šifrou nebo pevnou permutací, jako užitou komponentou, byly proto posuzovány zvlášť významně.

Přidané nástroje návrhu, jako zabudovaná pseudonáhodná funkce nebo mód pro tvorbu MAC kódů byly na konferenci také krátce diskutovány, ale při výběru finalistů nehráli příliš velkou roli.

Jako termín pro odevzdání vylepšených verzí algoritmu bylo stanoveno 15. září 2009. Změny byly oficiálně zveřejněny dva týdny poté, tak aby byl dostatek času tato vylepšení posoudit. Po tomto termínu některé soutěžící týmy ještě oznámily vylepšení, která by provedli na svých algoritmech, kdyby postoupili do finálního kola. NIST však již nebral při výběru finalistů tato vylepšení v úvahu.

V prosinci 2010 byli oficiálně oznámeni finalisté, jimiž jsou funkce BLAKE, Grøstl, JH, Kecaak a Skein.

V únoru 2011 byl vydán report, v němž NIST komentuje průběh výběrového řízení a uvádí důvody, proč jednotlivé funkce byly či nebyly vybrány mezi finalisty.

Trochu přinejmenším překvapující je, na což poukazuje i Klíma [64], výběr funkce JH.

Jak NIST v tomto reportu [89] uvádí, jeho cílem je dosáhnout bezpečnosti, resp. nalézt takové funkce, které budou bezpečné, nejméně na 20 následujících let, přičemž funkce by měli poskytovat stejnou úroveň bezpečnosti jako funkce definované v [84], což bylo mimo jiné jedním z hlavních kritérií. Krom toho jak, NIST tvrdí, nebral v úvahu vylepšení po výše uvedeném datu 15. září 2009. Tudíž je trochu zarážející, že funkce, na níž byl publikován útok snižující odolnost vůči nalezení vzoru pro 512 bitovou verzi, z 2^{512} operací na 2^{507}, postoupila do finále.

Útok samotný [15] byl publikován v roce 2010 a doporučuje zvýšení počtu kol zpracování z 35,5 na 42, aby byla bezpečnost zachována. Úprava JH byla uveřejněna v lednu 2011 [131], tedy až po vyhlášení finalistů.

Počet operací 2^{507} je jednoznačně vysoký, nicméně je již pod hranicí generického útoku a neodpovídá tak kritériím stanoveným samotným NIST.

Důvod výběru JH komentuje NIST takto: „JH byla vybrána jako finalista z důvodu jejího solidního bezpečnostního rozpětí, dobré celkové performanci a inovativnímu designu [89, str. 18]. „

Jednotlivé funkce byly pro třetí kolo opět vylepšeny. Vylepšení funkce JH již bylo zmíněno. U funkce BLAKE byly přejmenovány jednotlivé verze v souladu s SHA-2 a byl přidán počet kol zpracování kompresní funkcí (u verze 256 a 224 bitů byl počet zvednut na 14, u verze 512 a 384 bitů byl počet zvednut na 16) [2]. Funkce Grøstl byla vylepšena změnou posuvných hodnot uvnitř *Q* permutace a byly zvětšeny hodnoty konstant používaných v jednotlivých permutacích (*P* a *Q*) [47]. U funkce Skein byly upraveny posuny klíčů [118]. Úprava funkce Keccak spočívala v zjednodušení pravidla pro zarovnávání, byl odebrán parametr *d* a parametr *r* mohli tvořit pouze ty hodnoty jenž jsou násobkem osmi bitů [14]. Nyní tedy Keccak podporuje všechny hodnoty *r* rovné $0 < r \leq b$.

V období od 22. - 23. března 2012 byla ve Washingtonu D.C. konána poslední závěrečná konference, kde byly diskutovány finálové funkce. Zveřejnění výsledku soutěže by podle předběžného časového harmonogramu mělo proběhnout během současného čtvrtletí (duben – červen 2012).

Osobně si netroufám predikovat, kdo bude vítězem, nicméně na základě vyjádření NIST v [89] a samotné dokumentace se domnívám, že se bude patrně rozhodovat mezi BLAKE, Skein a Keccak.

5.3. Shrnutí

Veřejná soutěž NIST rozhodně velmi ovlivnila dění oblasti hašovacích funkcí a to v celosvětovém měřítku. Soutěžní týmy byly tvořeny odborníky doslova z celého světa, přičemž zde byla zastoupena i Česká republika a to nejen Vlastimilem Klímou (BMW), ale také Alešem Drápalem, profesorem Matematicko-fyzikální fakulty UK, který se podílel na tvorbě funkce Edon-R.

Lze říci, že výběr, který NIST učinil, byl opravdu velmi zodpovědný, jelikož z celkového počtu přihlášených funkcí, které postoupily do prvního kola, byla k současnosti prolomena více než polovina.

Funkce, které postoupily do druhého kola, se ukázaly jako velmi kvalitní a odolné, což dokazuje i skutečnost, že žádná z nich nebyla dosud prolomena, tedy alespoň ne zcela. Jak konstatoval Ronald Rivest [110], ani on sám nečekal takovou sílu útoků, jaká byla během této soutěže simulována, což vede k závěru, že i samotná kryptoanalýza v tomto oboru dosti pokročila.

Zajímavá je otázka, kterou řešil Bruce Schneier (mj. jeden z tvůrců Skein) na svém blogu, a sice, jak se v praxi bude nový standardní algoritmus nazývat. Označení SHA-2, jak bylo uvedeno v předchozí kapitole, je souhrnné neoficiální označení pro funkce s určitou definovanou délkou výstupu. Nový algoritmus by měl nést označení SHA-3 a poskytovat stejnou délku otisků jako SHA-2. Zároveň SHA-3 nemá SHA-2 nahradit, tedy alespoň ne prozatím. Jak tedy bude vypadat nové označení pro funkci z rodiny SHA s délkou výstupu např. 256 bitů, když SHA-256 již je definované. Tato skutečnost určitě není nijak zvlášť rozhodující, ale jako úvaha je opravdu zajímavá.

Funkce, které nebyly zvoleny do finále, jsou i nadále vyvíjeny a vylepšovány a je možné, že jednou budou v nějakém standardu zakotveny. Obecně to, že hašovací funkce není ukotvena v rámci nějakého standardu, neznamená, že v praxi není použitelná. Konkrétně například funkce Fugue, která je vyvíjena v laboratořích IBM, byla v současnosti upravena (2. dubna 2012) a byla publikována její nová verze [51].

Tím chci pouze demonstrovat skutečnost, že ačkoli soutěž NIST již prakticky skončila, vývoj na poli hašovacích funkcí neustále pokračuje.

6. Závěr

V rámci práce bylo stručně představeno celkem dvanáct teoretických konstrukcí hašovacích funkcí, tři konstrukce kompresní funkce a více než padesát realizovaných hašovacích funkcí. Minimálně okrajově byly zmíněny některé důležité projekty spojené s těmito funkcemi a také jejich standardizací.

Jak ukazuje historický vývoj, některé teoretické podklady pro tvorbu hašovacích funkcí přetrvaly dodnes. Všichni finalisté, potažmo semifinalisté soutěže NIST používají konstrukce, které lze považovat za vylepšenou variaci, případně následníka Merkle-Damgård. Což jen dokazuje význam kvalitních teoretických podkladů.

Cílem práce bylo podat celkový přehled o této problematice, což se, jak doufám, podařilo. Jednotlivé kapitoly tvořily určité samostatné okruhy a dílčí závěry lze nalézt ve shrnutí těchto kapitol. Některé skutečnosti byly rozebírány více či méně podrobně, ale domnívám se, že žádný ze závažných zlomů v této oblasti nebyl zcela opomenut a práce tak svého cíle dosáhla.

Použité zdroje

1. Aumasson, Jean-Philippe et all.. *SHA-3 proposal Blake*. In NIST submission package Round 1,2 [online]. Verze 1.2, 2008 [cit. 2012-04-10], 76 stran. Dostupné online také na: <http://ehash.iaik.tugraz.at/uploads/0/06/Blake.pdf>

2. Aumasson, Jean-Philippe et all.. *SHA-3 proposal Blake*. In NIST submission package Round 3 [online]. Verze 1.3, December 16, 2011 [cit. 2012-04-10], 79 stran. Dostupné online také na: <http://131002.net/blake/blake.pdf>

3. Anderson, Ross; Biham Eli. *Tiger: A fast new hash function*. In Fast Software Encryption: Lecture Notes in Computer Science. Springer:1996, Vol. 1039/1996, s. 89-97. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-60865-6_46>

4. Bagheri, Nasour; Naderi, Majid; Sadeghiyan, Babak. *Cryptanalysis of CRUSH hash structure*. Cryptology ePrint Archive, 043/2008 [cit. 2012-03-25], 8 stran. Dostupné online také na: <eprint.iacr.org/2008/043.pdf>

5. Barreto, Paulo S. L. M.. *The Whirlpool Hash Function* [online]. Laboratório de Arquitetura e Redes de Computadores: São Paulo, © 2001-2008, last update 2008.11.25. [cit. 2012-04-1]. Dostupný z: <http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html>

6. Bellare, Mihir; Rogaway, Phillip. *Random oracles are practical: a paradigm for designing efficient protocols*. In Proceedings of the 1st ACM conference on Computer and communications security. ACM Press,1993, s 62–73. Dostupné online také na: <http://charlotte.ucsd.edu/users/mihir/papers/ro.pdf >

7. Bellare, Mihir; Ristenpart, Thomas. *Multi-Property-Preserving Hash Domain Extention and the EMD Transform*. In Advances in Cryptology – ASIACRYPT 2006. Springer: 2006, s. 299-314. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/11935230_20>

8. Benadjila, R. et all.. *SHA-3 Proposal: ECHO*. In NIST submission package Round 2 [online]. Orange Labs, Verze 1.5, February 11, 2009 [cit. 2012-04-10], 50 stran. Dostupné online také na: <http://crypto.rd.francetelecom.com/echo/doc/echo_description_1-5.pdf >

9. Bentahar, Karnel et all.. *LASH*. In NIST – Second Cryptographic Hash Workshop, Santa Barbara, August 24-25, 2006. NIST, 2006 [cit. 2012-04-9], 10 stran. Dostupný také z: <http://csrc.nist.gov/groups/ST/hash/documents/SAARINEN_lash4-1_ORIG.pdf>

10. Bernstein, Daniel J.. *CubeHash specifications*. In NIST submission package Round 2 [online]. Verze 2.B.1, 2009 [cit. 2012-04-10], 5 stran. Dostupné online také na: <http://cubehash.cr.yp.to/submission2/spec.pdf >

11. Bertoni, Guido; Daemen, Joan; Assche, Gilles Van. *RadioGatún, a belt-and-mill hash function* [online]. In NIST – Second Cryptographic Hash Workshop, Santa Barbara, August 24-25, 2006. NIST, 2006 [cit. 2012-04-9], 15 stran. Dostupný také na: <http://csrc.nist.gov/groups/ST/hash/documents/VANASSCHE_RadioGatun_0720.pdf>

12. Bertoni, Guido et. all.. *Cryptographic sponge functions* [online]. In Ecrypt Hash Workshop, Barcelona, 2007. Rozšířená verze, 2011 [cit. 2012-04-9], 93 stran. Dostupné online také na: <http://sponge.noekeon.org/CSF-0.1.pdf>

13. Bertoni, Guido et all.. *Keccak Specifications*. In NIST submission package Round 2 [online]. Verze 2, September 10, 2009 [cit. 2012-04-10], 7 stran. Dostupné online také na: <http://keccak.noekeon.org/Keccak-specifications-2.pdf >

14. Bertoni, Guido et all.. *Keccak Specifications*. In NIST submission package Round 2 [online]. Verze 3, January 14, 2011 [cit. 2012-04-10], 14 stran. Dostupné online také na: <http://keccak.noekeon.org/Keccak-submission-3.pdf>

15. Bhattacharyya, Rishiraj; Mandal, Avradip; Nandi, Mridul. *Security Analysis of the Mode of JH Hash Function*. In Fast Software Encryption. Springer, 2010, 168-191. Dostupné online take na: <http://www.isical.ac.in/~rishi_r/FSE2010-146.pdf>

16. Biham, Eli; Shamir, Adi. *Differential Cryptanalysis of Feal and N-Hash*. In Advances in Cryptology-EUROCRYPT ´91. Springer: 1991, s, 1-16. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-46416-6_1>

17. Biham, Eli Shamir, Adi. *Differential Cryptanalysis of Snefru, Kafre, REDOC-II, LOKI and Lucifer*: Extended Abstract. Advances in Cryptology - CRYPTO´91. Springer: 1992, s. 156-171. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-46766-1_11>

18. Biham, Eli; Dunkelman, Orr. A Famework for Iterative Hash Functions – HAIFA [online]. In NIST – Second Cryptographic Hash Workshop, Santa Barbara, August 24-25, 2006. NIST, 2006 [cit. 2012-04-10], 9 stran. Dostupné také z: <http://csrc.nist.gov/groups/ST/hash/documents/DUNKELMAN_NIST3.pdf>

19. Biham, Eli. *New Techniques for Cryptanalysis of Hash Functions and Improved Attacks on Snefru*. Fast Software Encryption: Lecture Notes in Computer Science. Springer, 2008, Volume 5086, s 444-461. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-540-71039-4_28>

20. Biham, Eli; Dunkelman, Orr. *The SHAvite-3 Hash Function*. In NIST submission package Round 2 [online]. Vylepšená verze, 2009 [cit. 2012-04-10], 41 stran. Dostupné online také na: <http://www.cs.technion.ac.il/~orrd/SHAvite-3/Spec.15.09.09.pdf>

21. Boer, Bert den; Bosselears, Antoon. *An attack on the last two rounds of MD4*. In Advances in Cryptology - CRYPTO´91. Springer, 1992, s. 194-203. Dostupné online take na: <ftp://ftp.esat.kuleuven.ac.be/pub/COSIC/bosselae/md4rnd23.pdf>

22. Boer, Bert den; Bosselaers, Antoon. Collisions for the compression function of MD5. In Advances in Cryptology – EUROCRYPT ’93. Springer, 1994, s. 293 – 304. Dostupné online také na: <http://www.cosic.esat.kuleuven.be/publications/article-143.pdf>

23. Bosselears, Antoon; Govaerts, René; Vandewalle, Joos. *Cryptography Within Phase I of the EEC-RACE Programme*. In Computer Security and Industrial Cryptography. Springer, 1993, s. 227-234. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-57341-0_65>

24. Bosselears, Antoon. *RIPEMD Family*. In Encyklopedia of Cryptography and Security. Springer, 2011, Part 18, s. 1050-1053. Dostupné komerčně také ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-1-4419-5906-5_612>

25. Bresson, E. et all.. *Shabal, A Submission to NIST´s Cryptographic Hash Algorithm Competition*. In NIST submission package Round 1,2 [online]. Verze, October 28, 2008 [cit. 2012-04-10], 300 stran. Dostupné online také na: <http://ehash.iaik.tugraz.at/uploads/6/6c/Shabal.pdf>

26. Canniére, C. D., Sato, H., Watanabe, D.. *Hash Function Luffa*: *Specifiations*. In NIST submission package Round 2 [online]. Hitachi, Verze 2.0.1, October 2, 2009 [cit. 2012-04-10], 26 stran. Dostupné online také na: <http://www.sdl.hitachi.co.jp/crypto/luffa/Luffa_v2_Specification_20091002.pdf>

27. Contitni, S.; Lenstra, A.; Steinfeld R. *VSH, an efficient a provable collision-resistant hash function*. In Advances in Cryptology-EUROCRYPT 2006. Springer, 2006, s. 165-182. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/11761679_11>

28. Daemen, Joan; Clapp, Craig. *Fast Hashing and Stream Encryption with Panama*. In Fast Software Encryption. Springer, 1998, s. 60-74. Dostupné online také na: <http://web.eecs.utk.edu/~dunigan/cns04/panama.pdf >

29. Daemen, Joan; Assche, Giles Van. *Producing Collision for Panama, Instantaneously.* In Fast Software Encryption. Springer, 2007, s. 1-18. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-540-74619-5_1>

30. Daemen, Joan. *A Narrow Trail in Hash Functions Design*. In ESC 2008: Echternach Symetric Cryptography Seminar, 7.-11. ledna 2008. Université du Luxemburg: Echternach, 2008, 27 stran. Dostupné online také na: <http://wiki.uni.lu/esc/docs/ESC+Narrow+Trail+Hash.pdf>

31. Damgård, Ivan Bjerre. *A Design Priciple for Hash Function*. Advances in Cryptology-CRYPTO´89 Proceedings. Springer: 1990, s. 416-427. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/0-387-34805-0_39>

32. Damgård, Ivan Bjerre. *Collision free hash functions and public key signatures* schemes. Advances in Cryptology - EUROCRYPT´87. Springer: 1988, s. 203-216. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-39118-5_19>

33. Diffie, Whitfield; Hellman, Martin E.. *New directions in cryptography*. Information Theory, IEEE Transactions on. 1976,Vol. 22, Issue 6, s. 644-654. Dostupné online také na: <http://www-ee.stanford.edu/~hellman/publications/24.pdf > ISSN 0018-9448.

34. Dobbertin, Hans. *Alf Swindles Ann*. In CryptoBytes. RSA Laboratories, 1995, vol. 1, nr. 3., s. 5. Dostupné online:<ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto1n3.pdf>

35. Dobbertin, Hans. *RIPEMD with Two-Round Compress Function Is Not Collision-Free*. In Journalo of Cryptology. Springer, vol. 10, Nr. 1, 1997, s. 51-69. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/s001459900019>

36. Donghoon, Chang. *Preimage Attacks on CellHash, SubHash and Strengthened Version of CellHash and SubHash*. Cryprology ePrint Archive, 412/2006, 10 stran. [cit. 2012-04-01] Dostupné online také na: <http://eprint.iacr.org/2006/412.pdf>

37. Donghoon, Chang et all.. *Preimage Attack on the Parallel FFT- Hashing Function*. In Information Security and Privacy. Springer, 2007, s. 59-67. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-540-73458-1_5>

38. FIPS PUB 180-1. *Secure Hash Standard* [online]. National Institute of Standards and Technology, April 17, 1995. [cit. 2012-04-03]. Dostupný online také na: <http://www.itl.nist.gov/fipspubs/fip180-1.htm>

39. FIPS PUB 180-2. *Secure Hash Standard* [online]. National Institute of Standards and Technology, August 1, 2002. [cit. 2012-04-03]. Dostupný online také na: <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>

40. FIPS PUB 180-3. *Secure Hash Standard* [online]. National Institute of Standards and Technology, October, 2008. [cit. 2012-04-03]. Dostupný online také na: <http://csrc.nist.gov/publications/fips/fips180-4/fips180-4.pdf>

41. FIPS PUB 180-4. *Secure Hash Standard* [online]. National Institute of Standards and Technology, March, 2012. [cit. 2012-04-03]. Dostupný online také na: <http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf>

42. FIPS PUB 186-3. *Digital Signature Standard (DSS)* [online]. National Institute of Standards and Technology, June, 2009. [cit. 2012-04-07]. Dostupný online také na: <http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf>

43. FIPS PUB 198-1. *The Keyed-Hash Message Authentication Code (HMAC)* [online]. National Institute of Standards and Technology, June, 2009. [cit. 2012-04-07]. Dostupný online také na: <http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf>

44. Gauravaram, Praveen; Millan, William L.; May, Lauren J.. *CRUSH: A New Cryptographic Hash Function using Iterated Halving Technique.* In Cryptographic Algorithms and Their Uses. Goldcoast: QUT Publications, 2004, s. 28-39. Dostupné online také na: <saluc.engr.uconn.edu/refs/algorithms/hashalg/gauravaram04crush.pdf>

45. Gauravaram, Praveen et al.. *Constructing Secure Hash Functions by Enhancing Merkle-Damgård Construction.* In ACISP 2006, Lecture Notes in Coputer Science. Springer, 2006, Volume 4058, s. 407-420. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/11780656_34>

46. Gauravaram, Praveen et al.. *Grøstl A SHA-3 Candidate*. In NIST submission package Round 1, 2 [online]. Verze 1.0, October 31, 2008 [cit. 2012-04-10], 32 stran. Dostupné online také na: <http://groestl.info/Groestl-0.pdf>

47. Gauravaram, Praveen et al.. *Grøstl A SHA-3 Candidate*. In NIST submission package Round 3 [online]. Verze 2.0, March 2, 2011 [cit. 2012-04-10], 42 stran. Dostupné online také na:< http://www.groestl.info/Groestl.pdf>

48. Gligoroski, D.; Markovski, S.; Kocarev, L. Edon-R, *An Infinite Family of Cryptographic Hash Functions* [online]. In NIST – Second Cryptographic Hash Workshop, Santa Barbara, August 24-25, 2006. NIST, 2006 [cit. 2012-04-9], 9 stran. Dostupný také na: <http://csrc.nist.gov/groups/ST/hash/documents/GLIGOROSKI _EdonR-ver06.pdf>

49. Gligoroski, D. et all.. *Cryptographic Hash Function BLUE MIDNIGHT WISH*. NIST submission package Round 2 [online]. Verze, September, 2009 [cit. 2012-04-10], 77 stran. Dostupné online také na: <http://people.item.ntnu.no/~danilog/Hash/BMW-SecondRound/Supporting_Documentation/BlueMidnightWishDocumentation.pdf >

50. Halevi, S.; Hall, W. E.; Jutla, C. S.. *The Hash Function Fugue*. In NIST submission package Round 2 [online]. IBM,Verze, October 6 [cit. 2012-04-10], 2009, 97 stran. Dostupné online také na: <http://researcher.ibm.com/files/us-csjutla/fugue_Oct09.pdf >

51. Halevi, S.; Hall, W. E.; Jutla, C. S.. *The Hash Function Fugue*. In IBM Research [online]. IBM,Verze 2.0, April 2, 2012 [cit. 2012-04-10], 97 stran. Dostupné online také na: <http://researcher.ibm.com/files/us-csjutla/fugue_2012.pdf>

52. Han, Daewan; Park, Sangwoo; Chee, Seongtaeg. *Cryptanalysis of the Modified Version of the Hash Function Proposet at PKC´98*. Fast Software Encryption. Springer, 2002, s. 623-627. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-45661-9_20>

53. Henricksen, Matt; Kundsen, Lars. *Cryptanalysis of the CRUSH Hash Function.* In Lecture Notes in Computer Science. Springer, 2007, s. 74-83. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-540-77360-3_5>

54. Hirose, Shoichi; Park, Je Hong; Yun, Aram. *A Simple Variant of the Merkle-Damgård Scheme with a Permutation*. In Advances in Cryptology – ASIACRYPT 2007. Springer, 2007, s. 113-129. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/s00145-010-9095-5>

55. Hong, Dejuko et all.. *A New Dedicated 256-bit Hash Function: FORK-256* [online]. In NIST – First Cryptographic Hash Workshop, Gaithersburg, October 31- November 1, 2005. NIST, 2005 [cit. 2012-04-9], 15 stran. Dostupné online také na: <http://csrc.nist.gov/groups/ST/hash/documents/Sung_FORK.pdf >

56. ISO/IEC 10118-3:2004. Information technology -- Security Techniques -- Hash-functions -- Part 3: Dedicated hash-functions. ISO, 2004, 94 stran.

57. Jesang, Lee et all. *A New 256-bit Hash Function DHA-256: Enhancing the Security of SHA-256* [online]. In NIST – First Cryptographic Hash Workshop, Gaithersburg, October 31- November 1, 2005. NIST, 2005 [cit. 2012-04-9], 15 stran. Dostupné online také na: <http://csrc.nist.gov/groups/ST/hash/documents/ChangD_DHA256.pdf >

58. Joux, Antoine. *Multicollisions in iterated hash functions: Application to cascaded constructions*. In Advances in Cryptology – 2004. IACR, 2004, s. 306-316. Dostupné online také na:<www.iacr.org/cryptodb/data/paper.php?pubkey=1472>

59. Kahate, Atul. *Cryptography and Network Security*. New Dehli: Tara McGraw-Hill, ©2003,2008, 543 stran. Second edition. ISBN 0-07-064823-9.
ISBN 978-0-07-064823-4.

60. Kelsey, John; Schneier, Bruce. *Second preimages on n-bit hash function for much less than 2^n work*. In Advances in Cryptology – EUROCRYPT 2005. Springer, 2005, s. 474-490. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/11426639_28>

61. Klíma, Vlastimil. *Hašovací funkce, principy, příklady a kolize*. In Elekotronický Archiv článků Vlastimila Klími [online]. Klíma, verze 1, 19. 3. 2005. [cit. 2012-04-9] Dostupné online také na: <http://cryptography.hyperlink.cz/2005/cryptofest_2005.htm>

62. Klíma, Vlastimil. *Nalézání kolizí MD5 – hračka pro notebook*. In Elekotronický Archiv článků Vlastimila Klími [online]. Klíma, 5. března, 2005[cit. 2012-04-9], 6 stran. Dostupné online také na: <http://cryptography.hyperlink.cz/md5/MD5_kolize.pdf>

63. Klíma, Vlastimil. *Tunely v hašovacích funkcích: kolize MD5 do minuty*. In Elekotronický Archiv článků Vlastimila Klími [online]. Klíma, 2006[cit. 2012-04-9], 17 stran. Dostupné online také na: <http://cryptography.hyperlink.cz/2006/tunely.pdf>

64. Klíma, Vlastimil. Soutěž NIST SHA-3 a Blue Midnight Wish. In Personal pages: Vlastimil Klíma, Dr [online]. [cit. 2012-04-9] Dostupné online také na: <http://cryptography.hyperlink.cz/BMW/BMW_CZ.html>

65. Kundsen, Lars R.; Recheberger, Christian; Thomsen, Søren S. *The Grindhal Hash Functions*. In Lecture Notes in Computer Science. Springer, 2007, Volume 4593, s. 39-57. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-540-74619-5_3>

66. Küçük, Özgül. *The Hash Function Hamsi*. In NIST submission package Round 2 [online]. Verze, September 14, 2009 [cit. 2012-04-10], 20 stran. Dostupné online také na: <http://www.cosic.esat.kuleuven.be/publications/article-1203.pdf >

67. Leurent, Gaëtan. *MD4 is Not One-Way*. In Fast Software Encryption. Springer, 2008, s. 412-428. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-540-71039-4_26>

68. Leurent, G.; Bouillaguent, C.; Fougue, P.-A.. *SIMD is a Message Digest*. In NIST submission package Round 2 [online]. Verze, September 15, 2009 [cit. 2012-04-10], 270 stran. Dostupné online také na: <http://www.di.ens.fr/~leurent/files/SIMD.pdf >

69. Lucks, Stefan. *Design Principles for Iterated Hash Functions*. Cryptology ePrint Archive, 253/2004 [cit. 2012-03-24], 22 stran. Dostupné online: <http://eprint.iacr.org/2004/253.pdf>

70. Matyas, C.; Meyer, C.; Oseas, J.. *Generating strong one-way functions with cryptographic algorithm.* In Technical Disclosure Bulletin. IBM, vol. 3, 1985, s. 5658-5659. Dostupné komerčně online také na: <http://ip.com.//IPCOM/000063328>

71. Maurer, Ueli; Renner, Renato ; Holenstein, Clemens. *Indifferentiability, Impossibility Results on Redutions, and Applications to the Random Oracle Metodology*. In Theory of Cryptography. Springer, 2004, s. 21–39. Dostupné online také na: <http://eprint.iacr.org/2003/161.pdf >

72. Mendel, Florian; Pramstaller, Robert; Rechberger, Christian et al.. *Cryptanalysis of the GOST Hash Function.* In Advances in Cryptology – CRYPTO 2008. Springer, 2008, s. 162-178. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/0-378-3-540-85174-5_10>

73. Mendel, Florian; Rijmen, Vincent. *Colliding Message Pair for 53-step HAS-160*. Information Security and Cryptology - ICISC 2007. Springer, 2007, s. 324-334. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-540-76788-6_26>

74. Menzes, Alfred J.; Oorschot, Paul C. van; Vanstone, Scott A.. *Handbook of applied cryphography*. CRC Press, 1996. 816 stran. Dostupné online také na: <http://cacr.uwaterloo.ca/hac/> ISBN 0-8493-8523-7.

75. Merkle, Ralph C.. *Secrecy, athentication, and public key systems*. Ann Arbor:UMI Research Press, 1982. Disertační práce (PhD). Standford University, Department of Electrical Engineering,1979. Dostupné online také na www: <http://www.merkle.com/papers/Thesis1979.pdf> ISBN 0835713849.

76. Merkle, Ralph C.. *Certified Digital Signature: That Antique Paper from 1979*. In Advances in Cryptology - CRYPTO´89 Proceedings. Springer, 1990, s. 218-228. Dostupné také komerčně ze SpringerLink (DOI):< http://dx.doi.org/10.1007/0-387-34805-0_21>

77. Merkle, Ralph C.. *One Way Hash Functions and DES*. In Advances in Cryptology - CRYPTO´89 Proceedings. Springer,1990, s. 428-446. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/0-387-34805-0_40>

78. Microsoft Corporation. *Windows Server 2003: Internet Protocol Security for Microsoft Windows Server 2003* [online]. Microsoft Corporation, Version 1.2, 2/6/2008 [cit. 2012-03-24]. 21 stran. Dostupné online také na: <http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=12932>

79. Miyaguchi, S.; Ohta, K; Iwata,M. *128-bit Hash function (N-Hash).* In NTT review. JAPON, vol. 2, 1990, s. 128-132. ISSN 0915-2334.

80. Mridul, Nandi; Souradyuti, Paul. *Speeding Up The Wide-pipe: Secure and Fast Hashing* [online]. Cryprology ePrint Archive, 193/2010 [cit. 2012-03-24], 17 stran. Dostupné také na online: <http://eprint.iacr.org/2010/193.pdf>

81. MVČR. *Ministerstvo informovalo uživatele datových schránek o plánovaném přechodu na SHA-2* [online]. Tisková zpráva MVČR. In Tiskové zprávy 2010. Elektronický archiv tiskových zpráv MVĆR. MVČR, 14.5. 2010 [cit. 2012-03-29]. Dostupné online také na: <http://www.mvcr.cz/clanek/zpravodajstvi-ministerstvo-informovalo-uzivatele-datovych-schranek-o-planovanem-prechodu-na-sha-2.aspx>

82. Naito, Y. et all.. *Improved collision attack on MD4 with probability almost 1*. ICISC 2005: proceedings, Seoul, 1–2 December 2005. Springer, 2005, s. 129–145. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/11734727_12>

83. Naor, Moni; Yung, Moti. *Universal One-Way Hash Functions and their Cryptographic Applications*. In Poeceedigs of the twenty-first annual ACM symposium on Theory of computing. ACM, 1989, 33-43. Dostupné online také na: <www.wisdom.weizmann.ac.il./~naor/PAPERS/uowhf.ps>

84. National Institute of Standards and Technology. *Federal Register,* November 2, 2007. Vol. 72, No. 212. Dostupné onlina také na: <http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf>

85. National Institute of Standards and Technology. *NIST Comments on Cryptanalytic Attacks on SHA-1*. In Computer Security Division, Computer Security Resource Center [online]. NIST, April 25, 2006. [cit. 2012-04-04]. Dostupné online na: <http://csrc.nist.gov/groups/ST/hash/statement.html>

86. National Institute of Standards and Technology. *Tentative Timeline of the Development of New Hash Functions*. In Computer Security Division, Comptuer Security Resource Center [online]. NIST, July 12, 2006. [cit. 2012-04-04]. Dostupné online na: <http://csrc.nist.gov/groups/ST/hash/timeline.html>

87. National Institute of Standards and Technology. *Workshop Report: The Second Cryptographic Hash Workshop* [online]. Nechvatal, James; Chang, Shu-jen. University of California, Santa Barbara, August 24-25, 2006. NIST, 2006 [cit. 2012-04-04], 7 stran. Dostupné online také na: <http://csrc.nist.gov/groups/ST/hash/documents/SecondHashWshop%202006%20Report.pdf >

88. National Institute of Standards and Technology. *Workshop Report: The Firs Cryptographic Hash Workshop* [online]. Chang, Shu-jen; Dworkin, Morris. Gaithesburg, MD, Oct. 31-Nov. 1, 2005. NIST, 2005 [cit. 2012-04-04], 13 stran. Dostupné online také na: <http://csrc.nist.gov/groups/ST/hash/documents/HashWshop_2005_Report.pdf>

89. National Institute of Standards and Technology. *Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition* [online]. NIST Interagency Report 7764. NIST, February, 2011 [cit. 2012-04-04], 32 stran.. Dostupné online také na: <http://csrc.nist.gov/groups/ST/hash/sha3/Round2/documents/Round2_Report_NISTIR_7764.pdf>

90. Park, Nan Kynoung; Hwang, Joon Ho; Lee, Pil Joong. *HAS-V: A New Hash Function with Variable Output Length*. In Selected Areas in Cryptography. Springer 2001, s. 202-216. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-44983-3_15>

91. Peyrin, Thomas. *Cryptanalysis of Grindahl*. In Advances in Cryptology – ASIACRYPT 2007. Springer, 2007, s. 551-567. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-54076900-2_34>

92. PKCS #5 v2.1. *Password-Based Cryptography Standard* [online]. RSA Laboratories, October5, 2006 [cit. 2012-04-02]. Dostupný online také na: <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2_1.pdf>

93. Preneel, Bart. *Analysis and Design of Cryptographic Hash Functions*. Disertační práce (PhD). Katholieke Universitei Leuven,1993, 338 stran. Dostupné online také na www: <http://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf>

94. Preneel, Bart; Govaerts, René; Vandewalle, Joos. *Hash function based on block ciphers: a synthetic approach*. In Advances in Cryptology – CRYPTO´93. Springer, 1994, s. 368-378. Dostupné online také na: <ftp://www.zedz.net/pub/mirrors/Advances%20in%20Cryptology/HTML/PDF/C93/368.PDF>

95. Preneel, Bart. *Design principle for dedicated hash functions*. In Fast Software Encryption. Springer, 1994, s. 71-82. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-58108-1_10>

96. Preneel, Bart et all.. *Final report of European project IST-1999-12324, named New European Schemes for Signatures, Integrity, and Encryption*. Springer, Version 0.15 (beta), 2004, 829 stran. Dostupné online také na: <https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf >

97. Preneel, Bart. *The State of Hash Functions and the NIST SHA-3 Competition: Extended Abstract*. In Information Security and Cryptology. Springer, 2009, s. 1-11. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-642-01440-6_1>

98. Preneel, Bart. *Hash Functions*. In Encyclopedia of Cryptography and Security. Springer, 2011, Part 8, s. 543-553. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-1-4419-5906-5_580> ISBN 978-1-4419-5905-8. ISBN 978-1-4419-5907-2.

99. Quisquater, Jean J.; Girault Marc. *2n-Bit Hash Functions Using Symmetric Block Cipher Algorithms*. In Advances in Cryptology – Crypto´89. Springer, 1990, s. 102-109. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-46885-4_13>

100. Rabin, M. O.. *Digitalized Signatures*. In Foundations of secure computation. New York, Vol. 78, 1978, s. 155-166. Dostupné online také na: <http://smartech.gatech.edu/bitstream/handle/1853/40598/g-36-619_142482.pdf?sequence=1>

101. Rabin, M. O.. *Digitalized Signatures and public-key functions as intractable as factorization*. MIT, 1979, 20 stran. Dostupné online také na: <http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-212.pdf>

102. RFC 1319. *The MD2 Message-Digest Algorithm* [online]. Kaliski, B. S.. April 1992 [cit. 2012-04-02]. Dostupný z: <http://www.ietf.org/rfc/rfc1319.txt>

103. RFC 1320. *The MD4 Message-Digest Algorithm* [online]. Rivest, R.. April 1992 [cit. 2012-04-02]. Dostupný z: <http://www.ietf.org/rfc/rfc1320.txt>

104. RFC 1321. *The MD5Message-Digest Algorithm* [online]. Kaliski, B. S.. April 1992 [cit. 2012-04-02]. Dostupný z: <http://www.ietf.org/rfc/rfc1321.txt>

105. RFC 2104. *HMAC: Keyed-Hashing for Message Autentication* [online]. Krawczyk, H. et all.. February 1997 [cit. 2012-04-06]. Dostupný z: <http://www.ietf.org/rfc/rfc2104.txt>

106. RFC 3280. *Internet X.509 Public Key Infrastructure: Certificate and Certificate Revocation List (CRL) Profile* [online]. RSA Laboratories et. all, April 2002. [cit. 2012-04-06]. Dostupný z: <http://www.ietf.org/rfc/rfc3280.txt>

107. RFC 4122. *A Universally Unique Identifier (UUID) URN Namespace* [online]. Leach, P. et all.. July 2005 [cit. 2012-03-20]. Dostupný z: <http://www.ietf.org/rfc/rfc4122.txt>

108. RFC 5831. GOST R 34.11-94: *Hash Function Algorithm* [online]. Dolmatov, V Ed. March 2010 [cit. 2012-03-20]. Dostupný z: <http://tools.ietf.org/html/rcf5831> ISSN 2070-1721.

109. Rivest, Ronald L.. *The MD4 Message Digest Algorithm*. In Advaces in Cryptology-CRYPTO´90. Springer, 1991, s. 303-311. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-38424-3_22>

110. Rivest, Ronald L. *OFICIAL COMMENT: MD6*. In hash-forum@nist.gov [online]. NIST, Wed, 01 Jul 2009 10:41:45 -0400 [cit. 2012-04-03]. Dostupný také z www:<groups.csal.mit.edu/cis/md6/OFFICIAL_COMMENT_MD6_2009_07_01.txt>

111. Rjačko, Michal. *Properties of Cryptographic Hash Function*. In Cryprology ePrint Archive [online], 527/2008 [cit. 2012-04-04], 12 stran. Dostupné také online na: <http://eprint.iacr.org/2008/527.pdf>

112. Rogaway, Phillipe; Shrimptom Thomas. *Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Rreimage Resistace, Second-Preimage Resistance, and Collision Resistance*. In Fast Software Encryption: Lecture Notes in Computer Science 2004. Springer, 2004, Volume 3017/2004, s. 371-388. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-540-25937-4_24>

113. Sasaki, Yu. *Boomerang Distinguishers on MD4-Family: First Practical Results on Full 5-Pass HAVAL*. In Selected Areas in Cryptography. Springer, 2012, s. 1-18. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-642-28496-0_1>

114. Shin, Sang Uk et all.. *A new hash function based on MDx-family and its application to MAC*. In PUBLIC KEY CRYPTOGRAPHY. Springer, 1998, s. 234-246. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/BFb0054028>

115. Shirai, Taizo; Shibutani, Kyoji. *On the diffusion matrix employed in the Whirlpool hashing function.* In Public reports of the NESSIE project [online]. Katholieke Universitei Leuven, 2003 [cit. 2012-04-04]. Dostupné online také na: <http://www.cosic.esat.kuleuven.be./nessie/reports/phase2/whirlpool-20030311.pdf>

116. Schneier, Bruce. *Applied cryptography: Protocols, algorithms, and source code in C*. Second Edition. New York: John Wiley & Sons, 1996, 758 stran. Dostupné online také na: <ftp://ftp.eng.shirazu.ac.ir/Documents/Network%20Security/BOOKS/AC/APPLYCa.pdf> ISBN 0-471-12845-7. ISBN 0-471-11709-9.

117. Schneier, Bruce et all.. The *Skein Hash Function Family*. In NIST submission package Round 2 [online]. Verze 1.2, September 15, 2009 [cit. 2012-04-10], 92 stran. Dostupné online také na: <http://www.skein-hash.info/sites/default/files/skein1.2.pdf >

118. Schneier, Bruce et all.. The *Skein Hash Function Family*. In NIST submission package Round 2 [online]. Verze 1.3, October 10, 2010 [cit. 2012-04-10], 100 stran. Dostupné online také na: <http://www.skein-hash.info/sites/default/files/skein1.3.pdf>

119. Schneier, Bruce. *NIST Defines New Versions of SHA-512*. In Schneier on Secutity: A blog covering secutity and security technology [online]. Schneier, February 18, 2011[cit. 2012-04-03]. Dostupný online na: <http://www.schneier.com/blog/archives/2011/02/nist_defines_ne.html>

120. Schnorr, C. P. *FFT-II, Efficient Cryptographic Hashing*. In Advances in Cryptology - EUROCRYPT ´92. Springer, 92, s. 45-54. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-47555-9_4>

121. Schnorr, C. P.; Vaudenay, S.. *Parallel FFT-hashing*. In Fast Software Encryption. Springer, 1994, s. 149-156. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-58108-1_18>

122. Steinfeld, Ron et all.. *Cryptanalysis of LASH. In Fast Software Encryption*. Springer, 2008, s. 207-223. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/978-3-540-71039-4_13>

123. Vaudenay, Serge. *FFT-Hash II is not yet Collision-free*. In Advances in Cryptology – CRYPTO´92. Springer, 1993, s. 587-593. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-48071-4_43>

124. Vondruška, Pavel. *Hledá se náhrada za kolizní funkce…* . In Crypto-World. GCUCMP, 2006, ročník 8, číslo 5/2006, s. 2-5. Dostupné online na: <http://crypto-world.info/casop8/crypto05_06.pdf> ISSN 1801-2140.

125. Wang, X. et all.. *Cryptanalysis of the hash functions MD4 and RIPEMD*. In Advances in Cryptology – EUROCRYPT ’05. Springer, 2005, s. 1–18. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/11426639_1>

126. Wang, X.; Yu, H.. *How to break MD5 and other hash functions*. In Advances in Cryptology – EUROCRYPT ’05. Springer, 2005, s.19–35. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/11426639_2>

127. Wang, Xiaoyun; Yu, Hongbo; Yin, Yiqun Lisa. *Efficient Collision Search Attacks on SHA-0.* In Advances in Cryptology – CRYPTO´05. Springer, 2005, s.1-16. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/11535218_1>

128. Wang, Xiaoyun; Yin, Yiqun Lisa; Yu, Hongbo. *Finding Collisions in the Full SHA-1*. In Advances in Cryptology – CRYPTO´05. Springer, 2005, s.17-36. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/11535218_2>

129. Weisstein, Eric W. "Cryptographic Hash Function." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/CryptographicHashFunction.html

130. Wu, Hongjun. *The Hash Function JH*. In NIST submission package Round 1,2 [online]. Verze, September 15, 2009 [cit. 2012-04-10], 43 stran. Dostupné online také na: <http://ehash.iaik.tugraz.at/uploads/1/1d/Jh20090915.pdf>

131. Wu, Hongjun. *The Hash Function JH*. In NIST submission package Round 3[online]. Verze, January 16, 2011 [cit. 2012-04-10], 54 stran. Dostupné online také na: <http://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf>

132. Yuval, Gideon. *How to swindle Rabin*. In Cryptologia. Taylor & Francis, Vol. 3, Nr. 3, 1979, s. 187-191. Dostupné komerčně také z Taylor & Francis (DOI): <http://dx.doi.org/10.1080/0161-117991854025>

133. Zheng, Yuliang; Pieprzyk, Josef; Seberry, Jennifer. *HAVAL - A One-Way Hashing Algorithm with Variable Length of Output*. Advances in Cryptography – Auscrypt´92. Springer, 1993, s. 83-104. Dostupné také komerčně ze SpringerLink (DOI): <http://dx.doi.org/10.1007/3-540-57220-1_54>

Seznam vyobrazení

Obrázek 1 Vztahy mezi vlastnostmi hašovacích funkcí [112, s. 4]. 18
Obrázek 2 Kolize MD4 [34, s. 5] 19
Obrázek 3 Obecný model iterované hašovací funkce [74, s. 332] 27
Obrázek 4 Merkle-Damgårdova konstrukce [45, s. 409] 29
Obrázek 5 Wide-Pipe Hash [69, s. 8] 29
Obrázek 6 Double-Pipe Hash [69, s. 12] 30
Obrázek 7 Konstrukce EMD [7, s. 311] 31
Obrázek 8 Konstrukce 3C [45, s. 412] 32
Obrázek 9 Konstrukce 3CWP [45, s. 417] 32
Obrázek 10 Konstrukce 3C+ [45, s. 418] 33
Obrázek 11 Konstrukce MDP [54, s. 120] 33
Obrázek 12 Konstrukce Sponge [12, s. 13] 34
Obrázek 13 Konstrukce FWP [80, s. 3] 34
Obrázek 14 Konstrukce Matyas-Meyer-Oseas [74, s. 340] 35
Obrázek 15 Konstrukce Davies-Meyer [74, s. 340] 35
Obrázek 16 Konstrukce Miaguchi-Perneel [74, s. 340] 36