Slovenska technická univerzita
Fakulta elektrotechniky a informatiky

Hašovacie funkcie

Bakalárska práca

Michal Ondroš

FEI-5382-55806

2011

Slovenska technická univerzita
Fakulta elektrotechniky a informatiky
Ústav informatiky a matematiky

Hašovacie funkcie

Bakalárska práca

Michal Ondroš

FEI-5382-55806

Štúdijný program: Aplikovaná informatika
Štúdijný odbor: 9.2.9 Aplikovaná informatika
Školiace pracovisko: Ústav informatiky a matematiky
Vedúci bakalárskej práce: Mgr. Michal Mikuš

Bratislava, 2011

Zadanie

Štúdent: Michal Ondroš
Názov práce: Hašovacie funkcie
Štúdijný program: Aplikovaná informatika
Štúdijný odbor: 9.2.9 Aplikovaná informatika
Školiace pracovisko: Ústav informatiky a matematiky
Vedúci bakalárskej práce: Mgr. Michal Mikuš

V súčasnosti prebieha verejná súťaž NIST (National Institute of Standards and Technology) na výber novej hašovacej funkcie. Cieľom práce je urobiť prehľad zostávajúcich kandidátov a roztriediť ich do kategórií podľa použitých kryptografických primitív, resp. matematických štruktúr.

1. Naštudovať problematiku hašovacích funkcií
2. Vytvoriť všeobecný prehľad histórie hašovacích funkcií a ich využitie
3. Vyhodnotiť jednotlivé funkcie v bežiacej súťaži NIST a roztrieďte do vhodne zvolených kategórií

Anotácia

Slovenská technická univerzita v Bratislave
Fakulta elektrotechniky a informatiky

Autor:Michal Ondroš
Názov práce: Hašovacie funkcie
Vedúci práce: Mgr.Michal Mikuš
Dátum odovzdania: 13.5.2011
Kľúčové slová: Hašovacia funkcia, NIST, kandidát, finále, kompresná funkcia, konštrukcia hašovacej funkcie

V dnešnej digitálnej dobe, keď počítače využívame na každom kroku je nevyhnutné zabezpečiť ochranu súkromia ľudí, pri ich digitálnej komunikácii. Veľkú úlohu v tejto oblasti zohrávajú algoritmy známe ako hašovacie funkcie. Využívajú sa v digitálnych podpisoch, autorizačných protokoloch a mnohých ďalších oblastiach. Ich problémom ale je (podobne ako pri všetkých šifrách), že sa časom nájde nejaký sposôb ako prelomiť bezpečnosť takéhoto algoritmu. Preto v roku 2007 americký narodný inštitút štandardov a technológií (NIST) vyhlásil súťaž o nový štandard v oblasti hašovacích funkcií. V súčasnosti sa súťaž dostala do finále a v práci sa venujeme popisu kandidátov, ktorí postúpili do posledného kola súťaže a zároveň sme urobili experiment o rýchlosti jednotlivých návrhoch hašovacích funkcií.

Anotation

Slovak University of technology in Bratislava
Faculty of electrical engineering and information technology

Author:Michal Ondroš
Bachelor theses: Hash function
Supervisor: Mgr.Michal Mikuš
Date of submission: 13.5.2011
Key words: Hash functions, NIST, candidate, finale, compress function, construction of hash function

In our digital age, when we use computers on every step of our lives is necessary to secure the safety of the privacy people, in they digital communication. Hash algorithm play big role in that. They are used in digital signature, authorization protocols and many others fields. They problem is (like in all ciphers), that after some times, they will be discover some way to brake they. That is the reason, why American National Institute of Standards and Technologies (NIST) acclaim a company for new standard in hash algorithm. Now is the company in final and we describe the candidates, which advanced to final round and we make an experiment about speed the candidates.

Prehlasujem, že som túto bakalársku prácu na tému Hašovacie funkcie vypracoval samostatne, iba s pomocou literatúry uvedenej v zozname a konzultácií s vedúcim bakalárskej práce Mgr. Michalom Mikušom.

. . . . . .

Chcel by som sa poďakovať môjmu vedúcemu práce za veľmi ochotný prístup a pomoc pri písaní tejto práce. Takisto sa chcem poďakovať aj mojej rodine a priateľom za podporu.

Úvod

Ako už z názvu práce je zrejme, budeme sa venovať hašovacím funkciám. Po to čo boli prelomene hašovacie funkcie MD-5 a SHA-0 a SHA-1 sa ukázalo, že čím ďalej je viac dôležité brať na zreteľ bezpečnosť návrhov. Celá práca sa dá zameraním rozdeliť na dve hlavné časti. Prvú tvoria kapitoly v ktorých sa venujeme definíciám hašovacích funkcií, ich vlastnostiam, konštrukciám a útokom na ne vo všeobecnosti. Druhú časť tvorí popis jednotlivých kandidátov a experiment s nimi.

Úlohou tejto práce je priniesť prehľad o jednotlivých kandidátoch finálneho kola súťaže NIST o nový štandard v oblasti hašovacích funkcií. Snahou je, aby bol urobený čo najzrozumitelnejšie, aby každý kto si túto prácu prečíta si vedel vytvoriť obraz o jednotlivých návrhoch, ich vlastnostiach, vnútorných funkciách a bezpečnosti.

Obsah

Zoznam obrázkov

1 Úvodné definície

Skôr než sa začneme venovať jednotlivým kandidátom zo súťaže NIST, povieme si niekoľko teoretických vecí, ktoré budeme neskôr pri analýze potrebovať. V tejto časti si zadefinujeme základné veci okolo hašovacích funkcií.

Definície boli vytiahnuté z [MI1].

Hašovacou funkciou vo všeobecnosti rozumieme funkciu, ktorá je vo všeobecnosti založená na kompresnej funkcii a slúži väčšinou na autentifikáciu, digitálny podpis a podobne. Pre rovnaký vstup nám vždy musí vrátiť rovnaký výstup. Teda vo všeobecnosti to môžeme zapísať ako $h : D \rightarrow H$, kde $|D| > |H|$. Definičný obor D takejto funkcie tvorí množina všetkých správ, súborov, dokumentov ľubovoľnej konečnej dĺžky bitov a obor funkčných hodnôt budeme volať odtlačok alebo haš, pričom bude mať pevnú dĺžku n-bitov, pre každý typ hašovacej funkcie.

Hašovacie funkcie navyše môžeme rozdeliť podľa toho, či výsledný haš závisí len od vstupu MDCa samotnej funkcie, alebo sa tam pridáva aj nejaký tajný parameter, kľúč MAC.

Definícia 1.0.1. *Nech $D\{0,1\}^*aH\{0,1\}^m$, potom funkciu $h : D \rightarrow H$ budeme nazývať hašovacou ak pre každé M z D je ľahko vypočítať $h(M)$.*

Z tejto definície vyplíva, že množina vstupov je omnoho väčšia ako množina výstupov. Z toho logicky vyplýva, že musia existovať minimálne 2 rozdielne vstupy, ktoré budú mať rovnaký hash. Tento jav sa nazýva **kolízia**. Je samozrejme, že tomuto javu sa kvalitná hašovacia funkcia snaží vyhýbať. Na začiatok si ale určíme vlastnosti hašovacích funkcií.

Definícia 1.0.2. *Hašovacia funkcia $h : D \rightarrow H$sa nazýva jednosmernou ak pre $\forall y(H)$ je ťažké určiť $x(D)$.*

Táto vlastnosť nám zaručuje, že pokiaľ sa niekto dostane k odtlačku správy, je pre neho veľmi ťažké určiť pôvodnú správu. V kryptografii sa táto vlastnosť označuje ako "preimage resistance", teda v preklade odolnosť voči nájdeniu vzoru.

Definícia 1.0.3. *Hašovacia funkcia $h : D \rightarrow H$ má vlastnosť slabej odolnosti voči kolíziam ak pre $M(D) \rightarrow h(M)$ je ťažké nájsť $M\prime, M \neq M\prime$, kde $h(M) = h(M\prime)$.*

V anglickej literatúre sa tejto vlastnosti hovorí aj 2^{nd} preimage resistance, teda v preklade odolnosť vočí nájdeniu 2. vzoru. Táto vlasntosť nám hovorí o tom, že ak máme nejakú správu a k nej výsledný haš, musí byť ťažké vygenerovať inú správu, pre ktorú by bol vypočítaný rovnaký haš. Podobnou je aj nasledujúca vlasntosť, ktorá sa volá silná bezkolízovosť.

Definícia 1.0.4. *Hašovacia funkcia $h : D \rightarrow H$ má vlastnosť silnej odolnosti voči kolíziam ak je ťažké nájsť takú dvojicu $M_1, \neq M_2$, pre ktoré platí $M_1 \rightarrow h(M_1) \bigwedge M_2 \rightarrow h(M_1)$.*

To znamená, že musí byť ťažké nájsť dve rozličné správy, pre ktoré sa vypočíta rovnaký haš. Tieto dve vlastnosti sú nesmierne dôležité pri návrhu hašovacej funkcie. Väčšina útokov napádala práve túto vlastnosť. Ale k tomu sa podrobnejšie dostaneme neskôr.

V definíciách hore sme často spomínali pojem ťažko vypočítať, ťažko určiť. Tieto pojmy nie sú presne definované. Kedže ako množina vzorov tak množina hašov sú konečné množiny, teoreticky by sa dalo jednoducho prehľadať hrubou silou všetky možnosti. Nám ale bude stačiť, keď tento problém bude len prakticky ťažký, teda aby sa celé množiny nedali prehľadať ani na veľmi výkonných počítačoch v krátkom čase, resp. aby čas, ktorý by útočník musel do útoku investovať by už pre neho nebol zaujimavý.

Ďalšou vlastnosťou ktorú nebudeme matematicky definovať a musí ju hašovacia funkcia spĺňať je rovnomerné rozloženie výstupov, teda aby funkcia produkovala čo najmenej kolízií, tým pádom je ťažšie nájsť kolíziu a je aj odolnejšia voči útokom.

2 Typy útokov

Ako to už býva od nepamäti, ľudská zvedavosť chce vždy odhaliť to čo je skryté. Či už je to len z dlhej chvíle, alebo z nejakých zištných úmyslov, ekonomických, alebo preto aby dané tajomstvo mohol nejak inak zneužiť. Z tohoto dovôdu sa začala odveká "vojna" medzi kryptografmi a kryptoanalytikmi. Preto každá nová šifra, alebo kryptovací algoritmus je od momentu svojho zverejnia vystavený paľbe matematikov, elektrotechnikov, informatikov a vlastne kohokoľvek, kto sa pokusí daný algoritmus prelomiť. Keď svoju pozornosť upriamime len na hašovacie funkcie, tak zo všeobecného hľadiska by sa útoky dali rozdeliť do niektorých skupín.

Zo všeobecného hľadiska možeme podľa [MI1] cieľa deliť útoky na :

- **hľadanie vzoru správy** cieľom je nájdenie vzoru podľa výsledného hašu, teda nájsť pre h a $y \in R$ nájsť také M, pre ktoré bude platiť, že $h(M) = y$.
- **útok na druhý vzor** tento útok sa snaží zlomiť slabú odolnosť voči kolíziam, teda nájsť M, $h(M)$ také $M' \neq M$, pre ktoré by platilo, že $h(M') = h(M)$.
- **hľadanie kolízie** tento útok sa snaží prelomiť silnú odolnosť voči kolíziám, teda nájsť ľubovoľnú dvojicu M_1, M_2, pre ktorú bude platiť $h(M_1) = h(M_2)$.
- **útok na k-ty vzor** je to akási "nadstavba" hľadania vzoru správy. Snažíme sa nájst pre h a $y \in R$ nájsť také $M_1, M_2, \ldots, M_k$, pre ktoré bude platiť, že $y = h(M_1) = h(M_2) = \cdots = h(M_k)$.
- **hľadanie k kolízie** tu budeme hľadať k-ticu $M_1, M_2, ..., M_k$, pre ktorú bude platiť, že $h(M_1) = h(M_2) = \cdots = h(M_k)$.

Tvárme sa na chvíľu, že sme útočníci na istú hašovaciu funkciu, ktorú chceme prelomiť. Pokiaľ je naša situácia taká, že nevieme o akú hašovaciu funkciu presne ide, aký má typ konštrukcie a podobne tak sa nám možnosti útokov dosť zredukujú. Najzákladnejšou možnosťou je samozrejme prehľadať celú množinu vstupných reťazcov M a určite tam narazíme na kolíziu. Toto by bol ale časovo veľmi náročný problém. Druhou možnosťou je náhodné skúšanie rôznych dvojic. Tento spôsob by bol ale velmi neefektívny. Preto sa využíva tretia možnosť a tou je Birthday attack .

2.1 Birthday attack

Birthday attack (*Narodeninový útok*) je typ útoku na hašovaciu funkciu založený na hľadaní matematickej pravdepodobnosti o kolíziách. Pre jednoduchšie pochopenie problému si na začiatok uveďme príklad.

Predstavme si triedu v ktorej je 30 študentov. Každý z nich má svoj dátum narodenia, ktorý reprezentuje haš správy (Pre jednoduchšie počítanie budeme ignorovať 29. február). Takže možností dátumov narodení máme 365 s tým, že každý dátum má rovnakú pravdepodobnosť výskytu. Intuitívne a môže zdať, že pravdepodobnosť že dvaja študenti z tejto triedy majú rovnaký dátum narodenia je veľmi malá. Keby sme hľadali niekoho so špecifickým dátumom, tak pravdepodobnosť že nejaký študent je $1 - (364/365)^{30}$, teda približne 8 %. Keď by sme ale hľadali nejakého študenta s rovnakým dátumom ako ktorýkoľvek iný študent, tak pravdepodobnosť nájdenia nám narastie až na takmer 70% ($30 * 29/2 = 435$ párov študentov).

Vyberme z definičného oboru D náhodne n prvkov $(M_1, M_2, ..., M_n)$. Vypočítame $h(M_1), h(M_2)$atď, ktoré budú z oboru hodnôt H. Potom pravdepodobnosť, že nájdeme kolíziu sa rovná $p(n; H) \approx 1 - e^{-n(n-1)/(2H)} \approx 1 - e^{-n^2/(2H)}$.

Nech n je najmenší počet vyberaných prvkov. Potom invertovaním funkcie dostaneme $n(p; H) \approx \sqrt{}(2H \ln 1/(1-p))$.

Keďže pravdepodobnosť nemôže byť ľubovoľne malá, nastavíme ju na $0,5$. Potom dostaneme $n(0.5; H) \approx \sqrt{}((pi/2) * H)$.

Keď to naspať otočíme, výjde nám, že ak máme n-bitový kód, tak musíme vybrať $2^{n/2}$ prvkov aby sme našli kolíziu s pravdepodobnosťou $1/2$.

Keď si to zhrnieme, princíp útoku Birthday attack je vlastne veľmi jednoduchý.

1. Vyberieme n prvkov.
2. Vypočítame ich haš.
3. Vyhľadáme kolízie, ak sa nenašli opakujeme znova od bodu 1.

3 Typy konštrukcii hašovacích funkcii

V tejto časti si popíšeme základné konštrukcie hašovacích funkcií, na ktorých sú založení jednotliví kandidáti druhého kola súťaže NIST-u. Jedna sa o Merkle-Damgård konštrukciu, jej vylepšenia wide-pipe a HAIFA a konštrukciu Sponge. Na začiatok si ale popíšeme iterovanú konštrukciu, z ktorej vlastne všetky typy konštrukcii vychádzajú.

3.1 Iterovaná konštrukcia

Tento spôsob vytvárania hašovacích funkcií sa ukázal ako veľmi jednoduchý, efektný, že na jeho princípe pracuje dnes väčšina hašovacích funkcii. Iterovaná hašovacia funkcia spracováva správu po častiach. Označíme si dĺžku jednotlivých blokov b a predpokladajme, že správa M sa dá rozdeliť na presne L blokov $m_1, ..., m_L$. Potom spracovanie správy M možno realizovať pomocou funkcie $F : \{0,1\}^n \text{ x } \{0,1\}^b \rightarrow \{0,1\}^n$, ktorá už spracováva vstup konštantnej dĺžky. Výsledok i-teho kroku ide do $i+1$ kroku ako vstup spolu s $i+1$ časťou správy. Táto funkcia komprimuje správy veľkosti $(n+b)$ na správy veľkosti n bitov. Preto dostala označenie *kompresná funkcia*. Výstup funkcie F v i-tych krokoch, nazývame **medzivýsledky** a po spracovaní posledného bloku sa výsledok nazýva **výsledný odtlačok** správy M.

Tento návrh má ale ešte dva malé nedostatky. Prvým je to, že nie všetky správy sa pri pevnej dĺžke bloku daju rozdeliť bez toho aby posledný blok bol úplny. Teda bolo nutné dodefinovať ako zarovnať správu na L-násobok blokov. Najlepším spôsobom sa ukázalo pripísanie jednotky a za ňou toľko núl aby sme sa dostali k najbližšiemu možnému násobku dĺžky bloku.

Druhým problémom je, že v prvej iterácii nám chýba jeden vstup a to medzivýsledok z predchádzajúceho iterovania. Najjednoduchším spôsobom sa ukázalo prianie tzv. *inicializačného vektora* ako verejnej konštanty nultého medzivýsledku. Budeme ju označovať H_{IV} alebo H_0 .

Vo všeobecnosti môžeme definovať iterovanú hašovaciu funckiu h pomocou jej kompresnej funkcie F a spôsobom iterácie, ktorý sa skladá z inicializačného vektora a spôsobu zarovnania správy.

3.2 Merkle-Damgardova konštrukcia

Časom sa zistilo, že iterovaná konštrukcia má svoje bezpečnostné slabiny a ukázalo sa nevyhnutné vylepšenie tohto modelu. V roku 1989 prišli s vylepšením I. Damgård a nezávisle na ňom aj R. Merkle. Toto ich vylepšenie voláme Merkle-Damgårdova konštrukcia.

Princíp práce algoritmu je na začiatku rovnaký ako pri iterovanom modeli, správu si rozdelíme na L blokov a postupne prechádzajú kompresnou funkciou. Princíp vypešenia pri Merkle-Damgårdovom modeli spočíva v pridaní ešte jedného bloku na koniec, kde je binárne zapísaná dĺžka správy. Vďaka čomu silná bezkolízovosť závisí už len od kompresnej funkcie.

Toto vylepšenie sa ukázalo ako veľmi účinné, a veľa hašovacích funkci preto patrí do tejto triedy.

Pri vytváraní kompresnej funkcie sa najčastejšie používajú konštrikcie založené na blokových šifrách, ťažkých problémoch (napr. diskrétny logaritmus)alebo sú vytvorené špecialne tak, aby boli veľmi rýchle. V tomto prípade sa ale väčšinou nedá matematicky dokázať ich bezpečnosť.

3.3 Wide-pipe

V roku 2004 navrhol Stefan Lucas úpravu Merkle-Damgårdovej konštrukcie, ktorá ju štrukturálne vylepšuje. Ich cieľom bolo znemožnenie hľadania kolízií, pomocou znemožnenia hľadania kolízie kompresnej funkcie. Wide-pipe alebo aj jednoduché rozšírenie výstupu kompresnej funkcie na w bitov, kde $w > 2n$, je jedna možnosť ako to dosiahnuť. Pri tejto konštrukcii máme definované dve kompresné funkcie C_1 a C_2. $C_1 : \{0,1\}^w$ x $\{0,1\}^b \rightarrow \{0,1\}^w$ a $C_2 : \{0,1\}^w \rightarrow \{0,1\}^n$. Táto konštrukcia potom pracuje nasledovným spôsobom. Správa sa predspracuje klasickým Merkle-Damgårdovým spôsobom (rozdelenie na bloky, posledný s dĺžkou správy) a vypočíta sa odtlačok pomocou kompresnej funkcie C_1, ten sa po spracovaní posledného bloku spracuje pomocou funkcie C_2. Takto vznikne výsledný odtlačok.

3.4 HAIFA

Ďalším vylepšením Merkle-Damgårdovej konštrukcie je HAsh Iterative FrAmework, alebo skrátene HAIFA[HA]. Navrhli ju v roku 2006 E.Biham a O Dunkelman. Toto vylepšenie robí Merkle-Damgårdovu konštrukciu bezpečnejšou a variabilnejšou. Na rozdiel od wipe-pipe vylepšenia mení aj spôsob výplne a inicializačný vektor. Doterajšia kompresná funkcia používala na vstupe predchádzajúci medzivýsledok a blok správy, tu sa pridá ešte počet doteraz spracovaných bitov a reťazec, ktorý sa označuje ako "soľ". Formálny zápis je nasledovný: $F_{HA} : \{0,1\}^n x \{0,1\}^b x \{0,1\}^d x \{0,1\}^s \rightarrow \{0,1\}^n$, kde n je dĺžka medzivýsledku, b dĺžka bloku, d maximálna dĺžka správy a s dĺžka soli. Výpočet kompresnej funkcie je lineárny, tak ako pri Merkle-Damgårdovej konštrukcii.

Počet doteraz spracovaných bitov sa prikladá ako obrana proti útoku predlžovaním správy, pretože útočník už nemôže použiť $h(M)$ ako inicializačný vektor na výpočet $h(M\|y)$. Ak bol počet bitov správy M násobkom b, tak sa na záver spracovania $h(M)$ pridal blok výplne $10 \ldots 0$. Tento blok bol spracovaný kompresnou funkciou s rovnakým počtom spracovaných bitov, ako predchádzajúci blok. Ak by počet bitov M nebol násobkom b, tak by posledný výpočet odtlačku $h(M)$ obsahoval $\sharp bitov < b * i$. Pri výpočte $h(M\|y)$by bol posledný blok správy M spracovaný s počtom bitov rovným $b * i$ a to by vrátilo iný medzivýsledok kompresnej funkcie.

Pridanie soli do každého bloku správy znáhodní výpočet a znemožní akékoľvek predvýpočty daného dokumentu. Taktiež jednoduchou zmenou soli môžeme meniť bezpečnosť konštrukcie, čo zvyšuje jej variabilitu.

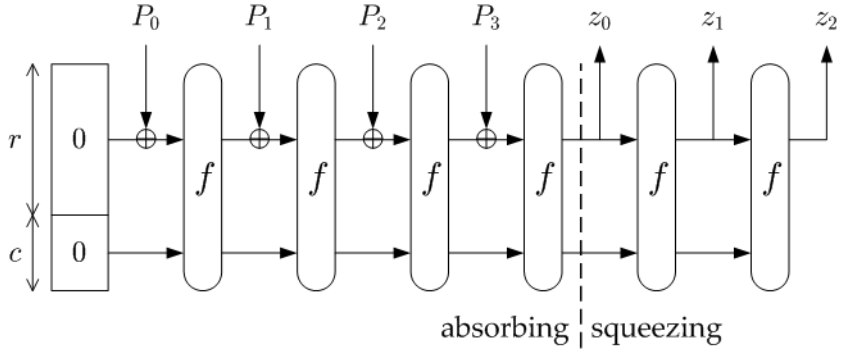
HAIFA podporuje ľubovoľné skrátenie dĺžky výsledného hašu. Pre každú dĺžku výstupu $n' < n$ je definovaný inicializačný vektor $IV_{n'}$, ktorý je vypočítavaný y globálneho inicializačného vektora. Výpočet prebehne s n-bitovou kompresnou funkciou a na konci sa skráti na požadovanú dĺžku. Vďaka rozdielnosti inicializačných vektorov, nebude kratší odtlačok prefixom dlhšieho.

Spôsob výplne sa definuje podobne ako pri Merkle-Damgårdovej triede. Doplní sa jednotka a potrebný počet núl. HAIFA je ľahko implementovateľná namiesto pôvodnej Merkle-Damgårdovej konštrukcie.

3.5 Sponge

Sponge [SP] je konštrukcia, ktorá pracuje zo vstupom voliteľnej dĺžky na výstup neobmedznej dĺžky. Kým si zadefinujeme samotnú konštrukciu ako funkciu, definujeme si niekoľko nevyhnutných vecí okolo nej:

- Nech $\mathbb{A}$ je grupa abecedy. Bude reprezentovať vstup a výstup. Grupa bude definovaná pomocou operácie sčítania a neutrálny prvok bude 0. V tomto kontexte môže reprezentovať bit, alebo aj n-bitov.
- Nech $\mathbb{C}$ je konečná množina, ktorá bude reprezentovať vnútornú časť stavu konštrukcie Sponge.
- Nech O je ľubovoľný prvok z množiny $\mathbb{C}$, ktorý bude časťou inicializačného vektora.
- Nech $p(m)$ je zobrazenie skupiny správ m na množinu reťazcov z grupy $\mathbb{A}$. Zobrazenie musí byť injektívne a musí platiť: $|p(m)| \geqslant 1$ a posledný znak $p(m)$ nesmie byť O.



Obrázok 1: Sponge

Sponge konštrukcia je funkcia, ktorá spracováva voliteľne dlhý vstup reťazca P znakov z grupy $\mathbb{A}$ na neobmedzene veľký výstup reťazca z znakov z grupy $\mathbb{A}$.Vstup je teda repzentovaný reťazcom znakov P, ktoré nekončia O a $|P| \geqslant 1$, jednotlivé znaky sú zapisované ako P_i; pre $0 \leqslant i < |P|$.

Konštrukcia Sponge má vnútorný stav $S = (S_A, S_C) \in \mathbb{A}X\mathbb{C}$, kde inicializačný vektor je $(0, O)$, pričom $0 \in \mathbb{A}$ a je neutrálnym prvkom grupy, $O \in \mathbb{C}$ je opísané hore.

Výpočet v Sponge prebieha v dvoch fázach:

- **Absorbing:** Pre každý vstupný znak P_i sa stav zmení na:

$$S \leftarrow f(S_A + P_i, S_C)$$

- **Squeezing:** Neobmedzený výstup z je zložený zo znakov $z_j \in \mathbb{A}$, kde

$$z_j = S_A$$

  a stav sa posunie na

$$S \leftarrow f(S)$$

Zadefinujeme si ešte dve hodnoty v konštrukcii:

- bitrate r: $r = \log_2 A, kdeA = |\mathbb{A}|$
- capacity c: $c = \log_2 C, kdeC = |\mathbb{C}|$

Konštrukcia Sponge operuje v rozsahu $b = r + c$ bitov, kde hodnota r sa volá bitrate a hodnota c kapacita.

Konštrukcia Sponge sa navyše ešte môže vylepšiť o takzvanú "Hermetic Sponge Strategy" založenej na základnej permutácii f, ktorá nemá žiadne štrukturálne rozlíšenia.

Konštrukcia Sponge ponúka unikátnu kombináciu výhod :

- Variabilnú dĺžku výstupu : môže generovať výstupy rôznej dĺžky a odteraz jedna funkcia môže byť použitá pre rôzne dĺžky výstupu.
- Založené na permutácii: základná funkcia f môže byť permutáciou lepšie ako kompresná funkcia alebo bloková šifra. Dizajnovanie vhodnej permutácie je jednoduchšie ako dizajnovanie vhodnej kompresnej funkcie alebo blokovej šifry.
- Odolnosť voči všeobecným útokom. Vďaka dokázanej indifferentibilite, pravdepodobnosť akéhokoľvek všeobecného útoku je ekvivalentná útoku na akékoľvek náhodné orákulum s dolným ohraničením $N^2 2^{-(c+1)}$, kde N je počet volaní funkcie f alebo f^{-1}.

Navyše konštrukcia Sponge , ktorá používa Hermetic Sponge Strategy má ešte nasledujúce výhody:

- Flexibilita: Používateľ si môže sám nastaviť pomer medzi rýchlosťou a bezpečnosťou jednoduchým určením adekvátneho bitratu a kapacity zatial čo používa stále tú istú permutáciu.
- Funkčnosť: Vďaka dĺžke výstupu a dokázanej bezpečnosti s rešektovaním všeobecných útokov, Sponge konštrukcia môže byť použitá priamo ako MAC funkcia, alebo prúdova šifra.

4 Kandidáti, 3. kolo

Dostávame sa do druhej časti práce, v ktorej sa už budeme venovať konkrétnym návrhom hašovacích funkcií. Budeme sa venovať ich konštrukcii, vnútorným funkciám a bezpečnosti. V priebehu písania práce bolo 5 kandidátov vybraných do finále. Sú nimi BLAKE, Grøstl, JH, Keccak a Skein.

4.1 BLAKE

Prvým finalistom súťaže NIST je BLAKE. Je to návrh založený na konštrukcii HAIFA. Vnútorná štruktúra je wide-pipe, ktorá používa hašovaciu funkciu LAKE. Kompresnou funkciou je modifikovaná prúdová šifra ChaCha[Cha1], ktorej bezpečnosť sa stále intenzívne skúma a má výborný výkon.

BLAKE je rodinou štyroch hašovacích funkcií: BLAKE-224, BLAKE-256, BLAKE-384, BLAKE-512, pričom prvé dve sú 32-bitové a druhé dve 64-bitové.

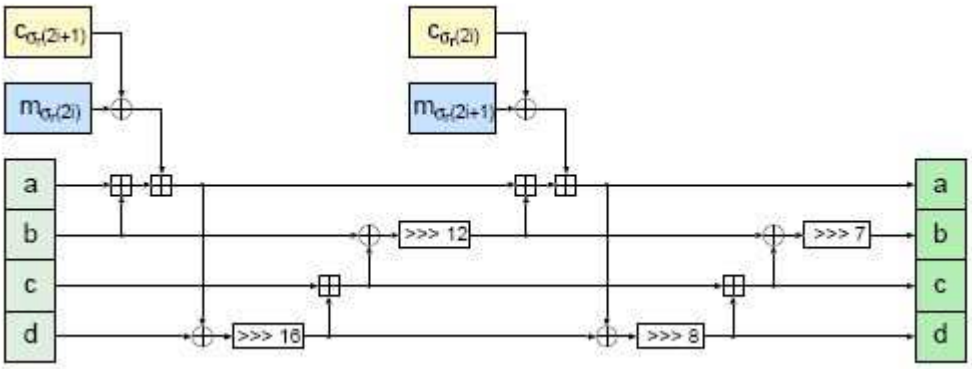
V skratke hašovacia funkcia BLAKE je založená na iterovanej konštrukcii HAIFA: kompresná funkcia závisí od soli a počtu už spracovaných bitov *counterovi*, na kompresiu každého bloku správy s rozdielnou funkciou. Štruktúra kompresnej funkcie BLAKE je kumulovaná funkciou LAKE[LA1]: veľký vnútorný stav je inicializovaný z počiatočnej hodnoty, soli a countera. Potom je injektivne upravený kolami ktoré závisia na správe a nakoniec kompresne spracovaný naspäť aby bol vstupom ďalšej časti zreťazenia. Táto stratégia sa volala lokálny wide-pipe a je inšpirovaná wide-pipe iterovaným módom.

Vnútorný stav kompresnej funkcie je reprezentovaný maticou slov veľkosti $4x4$. Jedno kolo BLAKE-256 je modifikované "dvojite kolo" prúdovej šifry ChaCha: Najskôr sa všetky 4 stĺpce nezavisle updatnu a potom štyri nedotýkajúce sa diagonály. Pri aktualizácii každého stĺpca alebo diagonály, dve slová správy sú vstupom zladenými do permutácie, ktorá zaleží od kola. Pre lepšiu predstavu si môžeme pozrieť nasledujúci obrázok. Každé kolo je parametrizované rozdielnou konštantou kôli minimalizovaniu podobnosti. Po sekvencii kôl,je stav redukovaný na polovicu svojej dĺžky so spätnou väzbou inicializačnej hodnoty a soli.

4.1.1 Elementárna analýza

Permutácie

Návrh využíva permutácie $\sigma_0 ... \sigma_9$, ktoré boli vybrané tak aby splnené niektoré bezpečnostné kritéria. Najskôr sa zabezpečilo, aby sa vstupný rozdiel neukázal dvakrát na tom istom mieste. Potom aby sa pre náhodnu



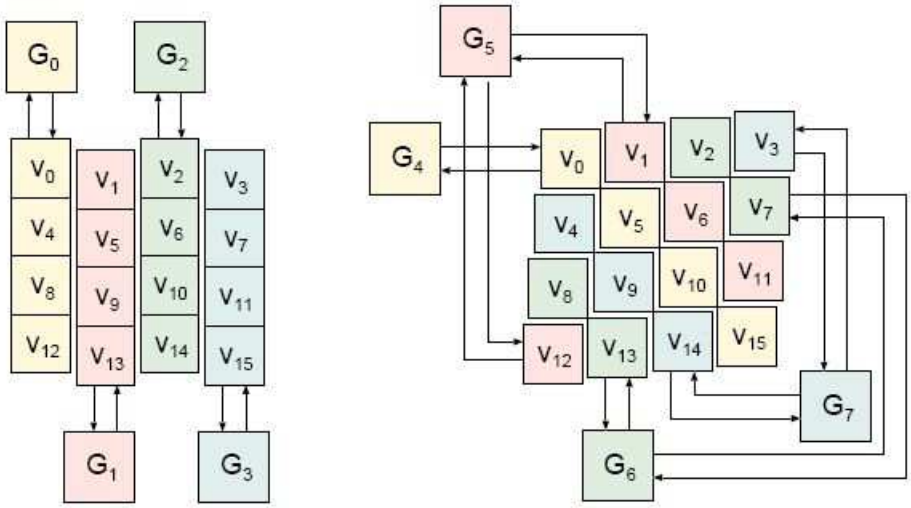
Obrázok 2: stĺpcové permutácie

správu všetky hodnoty $(m_{\sigma_r 2i} \bigoplus c_{\sigma_r 2i+1})$ a $(m_{\sigma_r 2i+1} \bigoplus c_{\sigma_r 2i})$ boli rozdielne s vysokou pravdepodobnosťou. Pre vybrané správy to garantuje, že každé slovo správy bude XOR-ované s inou konštantou, čo zaručí rozdielnosť transformácii na pozície cez kolá. Taktiež to implikuje že žiadna dvojica (m_i, m_j) nebude dvakrát vstupom v rovnakom G_i. Nakoniec pozícia vstupov by mala byť balancovaná: v jednom kole je daná správa vstupom buď pre stĺpocovú časť alebo pre diagonálnu časť a zaručuje tiež prvú alebo druhú pozíciu vo výpočte G_i. Je zaručené, že každá slovo správy je práve toľkokrát v stĺpcovej ako diagonálnej časti a práve toľkokrát na prvom mieste ako na druhom mieste vo výpočte G_i. Ukážku môžeme vidieť v nasledujúcej tabuľke.

kolo	G_0		G_1		G_2		G_3		G_4		G_5		G_6		G_7	
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	14	10	4	8	9	15	13	6	1	13	0	2	11	7	5	3
2	11	8	12	0	5	2	15	13	10	14	3	6	7	1	9	4
3	7	9	3	1	13	12	11	14	2	6	5	10	4	0	15	8
4	9	0	5	7	2	4	10	15	14	1	11	12	6	8	3	13
5	2	12	3	10	0	11	8	3	4	13	7	5	15	14	1	9
6	12	5	1	15	14	13	4	10	0	7	6	3	9	2	8	11
7	13	11	7	14	12	1	3	9	5	0	15	4	8	6	2	10
8	6	15	14	9	11	3	0	8	12	2	13	7	1	4	10	5
9	10	2	8	4	7	6	1	5	15	11	9	14	3	12	13	0

G funkcia

Dané (a, b, c, d) a bloky správy $m_j, j \in 0, ..., 15$ počíta
$a \leftarrow a + b + (m_{\sigma_r 2i} \bigoplus c_{\sigma_r 2i+1})$
$d \leftarrow (d \bigoplus a) >>> 16$



Obrázok 3: diagonálové permutácie

$c \leftarrow c + d$
$b \leftarrow (b \bigoplus c) >>> 12$
$a \leftarrow a + b + (m_{\sigma_r 2i+1} \bigoplus c_{\sigma_r 2i})$
$d \leftarrow (d \bigoplus a) >>> 8$
$c \leftarrow c + d$
$b \leftarrow (b \bigoplus c) >>> 7$

Kompresná funkcia vychádza z funkcie prúdovej šifry ChaCha, konkrétne z funkcie "quarter-round". Rozdiel je v pridaní dvoch slov správy (m) a konštánt (c). Rotácie boli ponechané z ChaCha-e, ktoré dávajú dobrý pomer bezpečnosť/efektívnosť. 16 a 8 bitové rotáciu sú rýchle na 8-bitových procesoroch, pretože ide o posun o bajty a teda nie sú potrebné žiadné rotačné inštrukcie. Bitové rotácie veľkosti 12 a 7 zasa lámu bajtovu štruktúru a sú aj relatívne rýchle.

Funkcie počas kôl

Funkcie, ktoré sa používajú počas jednotlivých kôl výpočtu sú:

$G_0(v_0, v_4, v_8, v_{12}), G_1(v_1, v_5, v_9, v_{13}), G_2(v_2, v_6, v_{10}, v_{14}), G_3(v_3, v_7, v_{11}, v_{15}),$
$G_4(v_0, v_5, v_{10}, v_{15}), G_5(v_1, v_6, v_{11}, v_{12}), G_6(v_2, v_7, v_8, v_{13}), G_7(v_3, v_4, v_9, v_{14})$

Pretože G je permutácia, jedno kolo je permutácia skrytého stavu v pre akúkoľvek pevnú správu. Inými slovami, daná správa a hodnota v po r kolách, a niekto môže určiť hodnotu v po $r-1, r-2, ...$ kolách a tak zistiť

inicializačnú hodnotu. To znamená, že nikto nemôže nájsť dve správy, ktoré inicializujú rovnakú počiatočnú hodnotu po akomkoľvek počte kôl. Takto je zabezpečená difúzia funkcie.

4.1.2 Hašovacia funkcia

Hašovacia funkcia BLAKE sa skladá z troch častí, ktoré využívajú funkcie, ktoré sme spomínali v predchádzajúcej časti inicializačnej funkcie, funkcie počas kôl a finalizácie. Poďme si pozrieť jednolivé časti.

Inicializácia

Na inicializačnom stupni, konštanty a redundancia t sa uloží do nenulového stavu (a tiež nie do stavu samých jednotiek). Rozdelenie vstupov naznačuje, že po prvom stĺpcovom kole je inicializačná hodnota h okamžite premiešaná so soľou s a counter-om t.

Dvojitý vstup t_0 a t_1 pri inicializačnom stave naznačuje predstavu hodnoty inicializačného stavu: Môžeme zavolať inicializačný stav $v_0, ..., v_{15}$ iba ak existujú t_0, t_1 tak potom $v_{12} = t_0 \oplus c_4$ a $v_{13} = t_0 \oplus c_5$ a $v_{14} = t_1 \oplus c_6$ a $v_{15} = t_1 \oplus c_7$. Stavy bez hodnôt nie je možné inicializovať.

Počet kôl

Pre finálnu verziu, boli určené nasledujúce počty kôl: 14 pre BLAKE-224 a BLAKE-256, 16 pre BLAKE-384 a BLAKE-512.

Výber viac konzervatívneho prístupu k bezpečnosti bol zdôvodnený implementáciou a výsledkami kryptoanalýzy, ktoré boli publikované v decembri roku 2010.

Finalizácia

Na finalizačnom stupni je stav kompresovany na polovicu jeho dĺžky, podobným spôsobom ako šifra Rabbit[RA1]. Spätná väzba h a s robia každé slovo hašu závislé na dvoch slovách v rozdielnych stavoch, jedno slovo inicializačnej hodnoty a jedno slovo zo soli. Zámerom je urobiť funkciu tak, aby nebola invertibilna v prípade že inicializačná hodnota a/alebo hodnota soli sú neznáme.

Lokálne kolízie

K lokálnej kolízii dochádza, keď pre dve rozdielne správy je stav po určitom počte kôl rovnaký. Pre hašovaciu funkciu BLAKE, neexistujú žiadne lokálne kolízie pre nejaký inicializačný stav (napr. rovnaká inicializačná hodnota, soľ, alebo counter). Tento výsledok priamo vychádza z faktu, že funkcia

v kole je permutácia správy pre pevný inicializačný stav v (a tak rôzne vstupy dajú rôzne výstupy). Táto schopnosť platí pre akýkoľvek počet kôl. Potreba rovnakého inicializačného stavu príliš neovplyvňuje výsledok: pre väčšinu aplikácii, sa nepoužíva žiadna soľ a kolízia hašovacej funkcie implikuje kolíziu na kompresnej funkcii s rovnakým inicializačným stavom.

4.1.3 Bezpečnosť

Pre všetky hašovacie funkcie BLAKE, by podľa výrobcu nemal existovať podstatne efektnejší spôsob ako štandartné metódy hrubej sily pre hľadanie kolízií s rovnakou alebo rozdielnou soľou, hľadanie silnej kolízie s ľubovoľnou soľou. BLAKE by ale mal byť bezpečný proti náhodnému hašovaniu. Podľa aktuálnych výsledkov kryptoanalýzy, ktoré ma NIST kdispozícii je nemožné rozlišovať priklad BLAKE-u pri neznámej soli (ktorá je vždy náhodne vybraná z PRF), dané prístupom do blackboxu funkcie presnejšie, nemalo by byť možné významne menej ako ako $2^{|s|}$ otázok na box kde $|s|$ znamená veľkosť soli. Podľa výsledkov kryptoanalýzy v prvých dvoch kolách BLAKE nemá žiadne vlastnosti, ktoré by ho robili významne menej bezpečným, ako ideálna funkcia pre ktorúkoľvek konkrétnu aplikáciu. (zahŕňa to podmienku, že funkcia musí mať odporúčaný počet kôl, nesmie byť nijako redukovaná alebo modifikovaná).

Všeobecné útoky
Útok predlžovaním správy[CR10] Útok predlžovaním správy nie je možný na návrh BLAKE, pretože rozdiel medzi pôvodnou správou a jej predĺžením je príliš rozdielny a nedá sa medzi nimi nájsť súvislosť.

Jouxov útok[JO04] Tento útok nie je reálnou hrozbou a to preto, že keby mal byť, tak by sme museli poznať aspoň dve kolízie, ale ktorákoľvek funkcia, pre ktorú by sa našla bola by prelomená tak či tak. Preto je tento útok možný len voči hašovacím funkciám, ktoré nie sú odolné voči kolíziám.

Birthday attack Birthday attack na permutáciu v nebude úspešný s pravdepodnosťou vyššou ako 2^{-32} (pre BLAKE-256). Takže pri dodržaní odporúčaného počtu kôl je podľa výrobcu prakticky nepoužiteľný.

4.1.4 Zhrnutie

Zhrňme si teda výhody a nevýhody BLAKE:

VÝHODY

dizajn

- jednoduchosť algoritmu
- rozhranie pre hašovanie so soľou

výkon

- rýchly na harvéri aj softvéri
- pararelnosť a prechodový kompromis pre hardvérovú implementáciu
- jednoduchý kompromis medzi bezpečnosťou a rýchloťou pomocou nastaviteľného počtu kôl

bezpečnosť

- založený na intezivne skúmanom komponente *ChaCha*
- odolné voči všeobecným útokom na silnú bezkolízovosť
- odolné voči útokom postrannými kanálmi
- odolné voči útokom predlžovaním

NEDOSTATKY

- Dĺžka správy obmedzená na 2^{64} $preBLAKE - 256$ respektíve 2^{128} $preBLAKE - 512$
- Odolnosť voči Jouxovým multikolíziám podobná ako má SHA-2
- Nájdenie pevných bodov v kratšom čase ako pri idealnej funkcii (ale nie efektívne)

4.2 Grøstl

Druhým kandidátom, ktorý sa prebojoval do finale je Grøstl. Za jeho tvorbou stojí tím z Dánskej technickej univerzity a TU Graz.

Grøstl je iterovaná hašovacia konštrukcia, kde je kompresná funkcia založená na dvoch širokých rozdielnych permutáciách. Dizajn Grøstl je transparentný a jeho princípy sú veľmi rozdielne od princípov rodiny SHA.

Tieto permutácie, ktoré sa používaju pri kompresii používaju "wide trail desing" [DA01] stratégiu, ktorá dáva Grøstlu možnosť odolávať veľkému množstvu kryptoanalytických útokov. Navyše ak sa permutáciu dajú ako ideálne, dá sa dokázať bezpečnosť hašovacej funkcie.

Grøstl je bajtovo-orienotovaný SP-network, ktorý vyžíva niektoré kompenenty z AES [DA02]. S-box je rovnaký, aký sa používal na blokové šifry AES a difúzne vrstvy sú konštruované podobným spôsobom ako sú tie v AES. Spoločne vytvárajú veľmi dobré konfúzne a difúzne vlastnosti modelu.

Grøstl je so called wide-pipe konštrukcia, kde je veľkosť vnútorného stavu omnoho väčšia ako veľkosť výstupu. To spôsobuje, že generické útoky sa stávaju omnoho zložitejšie a náročnejšie.

Grøstl má dobrý výkon na širokom spektre platforiem a obrana proti útokom postrannými kanálmi je veľmi dobrá vďaka jednoduchej práci AES.

4.2.1 Elementárna analýza

Grøstl je skupinou štyroch hašovacích funkcií, podľa dĺžky výsledného hašu sú to Grøstl-224, Grøstl-256, Grøstl-384 a Grøstl-512. V nasledujúcej časti si rozoberieme funkcie, ktoré využíva.

Konštrukcia hašovacej funkcie

Hašovacia funkcia Grøstl iterovane spracúva správu pomocou kompresnej funkcie f nasledovne: Správa M je rozdelená na bloky $m_1...m_t$ veľkosti l-bitov a bloky sú spracovávané jeden po druhom. Inicializačné hodnoty sú samostatne definované a pevne nastavené a ostatné bloky správ sú potom spracované ako:

$$h_i \leftarrow f(h_{i-1}, m_i) \text{ pre } i = 1, 2...t$$

Teda, funkcia f spracováva 2 vstupy veľkosti l na jeden výstup veľkosti l. Podrobnejšie sa tejto kompresnej funkcii bude venovať nižšie. Pre jednotlivé verzie je l dané na 512 bitov, pre verzie s dĺžkou hašu menšou ako 256 bitov. A 1024 bitov pre dlhšie. Potom čo sú bloky správ spracované, je výsledný haš H_M vypočítaný ako :

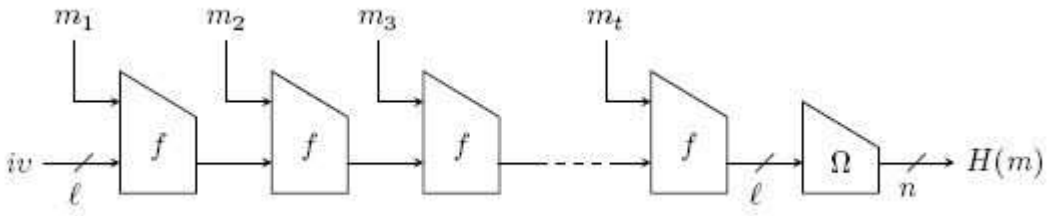
$$H_M = \Omega(h)_t$$

kde Ω je definovaná ako výstupná transformácia. Jej sa ale budeme venovať až v ďalšej časti.

Schému návrhu si môžeme pozrieť na nasledujúcom obrázku.

Permutácie P a Q

Skôr, ako sa dostaneme k samotnej kompresnej funkcii f, musíme si ešte popísať permutácie, ktoré využíva. Majú dve varianty, pracujú s dĺžkou 512 alebo 1024 bitov. Rozdiel je v princípe len vo veľkosti tabuľky, s ktorou pracujú. V princípe ale pracujú rovnako, či už ide o krátku alebo dlhú verziu.



Obrázok 4: návrh grostl f

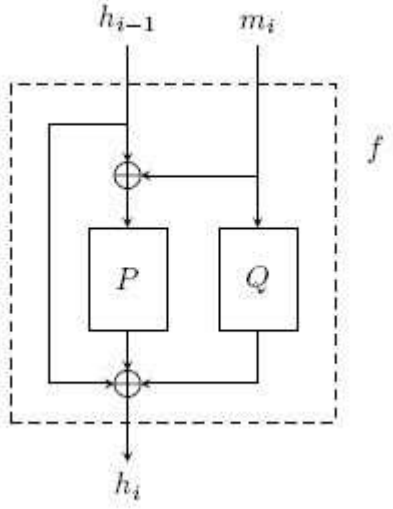
Dizajn je inšpirovaný blokovou šifrou Rijandael, o ktorej sa môžeme dočítať na [RI01] a [RI02]. Z toho vyplýva, že návrh pozostáva z počtu kôl R, pričom v každom sa používajú funkcie, ktoré využívajú *pridanie konštanty do poľa tabuľky, nahradenie bajtu, bajtový posun, výmena bajtov.* Permutácie využívaju poľe veľkosti 8x8, v prípade dlhších verzií je to pole veľkosti 8x16.

Kompresná funkcia f

Kompresná funkcia f je založená na dvoch l-bitových permutáciách P a Q. Je definovaná nasledovne:

$$f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h$$

Funkciu f vidíme aj na nasledujúcom obrázku:



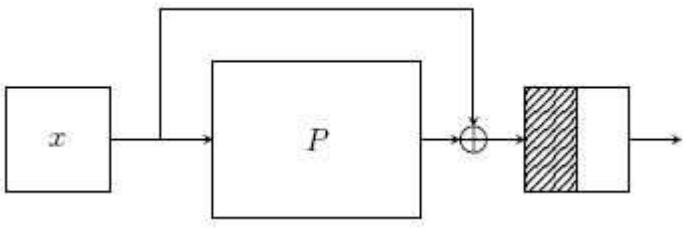
Obrázok 5: kompresná funkcia f

Výstupná transformácia Ω

Nech $trunc_n(x)$ je definovaná ako funkcia, ktorá odstraňuje všetko až na posledných n bitov zo správy dĺžky x. Potom funkcia ω je definovaná:

$$\omega(x) = trunc_n(P_x \oplus x).$$

Funkciu ω môžeme vidieť aj na nasledujúcom obrázku.



Obrázok 6: výstupná funkcia Ω

Počet kôl

Počet kôl r je nastaviteľný bezpečnostný parameter. Odporúčaných je 10 kôl pre krátke verzie a 14 pre dlhé verzie.

4.2.2 Bezpečnosť

Kompresná funkcia f bola označená ako bezpečná, pretože permutácie P a Q, ktoré využíva sú ideálne, podľa [FO08]. Bezpečnostný dôkaz hovorí, že je potrebných najmenej $2^{l/4}$ vyhodnotení P a/alebo Q aby bola nájdená kolízia pre hašovaciu funkciu aby opakovala f a že je najmenej $2^{l/2}$ vyhodnotení potrebných na nájdenie vzoru. To ukazuje, že hodnoty sú druhou odmocninou bezpčnosti ideálnej kompresnej funkcie. Každopádne od $l \geqslant 2n$, hodnoty sa zvyšujú na $2^{n/2}$ a 2^n. Navyše táto analýza nezohľadňuje výstupnú funkciu Ω.

Hľadanie kolízií v kompresnej funkcii

Všeobecný birthday attack na kompresnú funkciu má zložitosť $2^{n/3}$ a teda je rýchlejší ako útok na ideálnu kompresnú funkciu. Treba poznamenať, že je stále nad úrovňou spomýnaných $2^{l/4}$ a je nad hodnotou zložitosti birthday attacku na hašovaciu funkciu pre $n \leqslant l/2$. Útok nedáva priestor útočníkovi príliš kontrolovať hodnoty vstupu a teda výrobcovia nevidia žiadnu možnosť rozšíriť tento útok na celú hašovaciu funkciu.

Hľadanie kolízií v hašovacej funkcii Návrh konštrukcie je preukázateľne odolný voči nájdeniu kolízii až do $2^{l/4}$ volaní permutácii. Stále, žiaden kolízny

útok takejto zložitosti nie je známy, keď permutáciu sú označené ako ideálne. Najlepší známy kolízny útok potrebuje $2_{3l/8}$ volaní permutácie, ale naozajstná zložitosť v časti volania porovnateľnej kompresnej funkcie je vyššia ako $2^{l/2}$. Teda, stále je vysoká miera bezpečnosti v návrhu.

Jouxov útok Multikolízny Jouxov útok aplikovaný na návrh Grøstl: zložitosť nájdenia k-tej kolízie je $\log_2(k)2^{l/2} \geqslant \log_2(k)2^n$. Toto by sa dalo porovnať s hľadaním multikolízií pomocou metódy úplného prehľadávania hašovacej funkcie, kde zložitosť je okolo $(k!)^{1/k} * 2^{n(k-1)/k}$. Pre kryptograficky relevantné hodnoty k a n je útok úplným prehľadávaním vždy rýchlejší ako Jouxova metóda.

Útok predlžovaním správy Útok predlžovaním správy nie je jednoduché uskutočniť na tomto návrhu, ibaže by správy kolidovali ešte pred výstupnou transformáciou. Nájdenie kolízie pred výstupnou transformáciou trvá $2^{l/2} \geqslant 2^n$ pomocou birthday attacku. Ako je spomenuté niekedy sa vynecháva útok na hašovaciu funkciu s výstupnou transformáciou, ale výrobcom, nie je známy žiaden útok so zložitosťou nižšou ako birthday attack.

4.2.3 Výhody

- Jednoduchá analýza, keďže Grøstl je založený na malom počte permutácií, na rozdiel od blokových šifier.
- Bezpečná konštrukcia, výrobcovia predpokladajú ideálne permutácie.
- Veľmi známy dizajn založený na permutáciách.
- Nešpecializované požiadavky presnej platformy, výrobca udáva dobrý výkon na širokom spektre platforiem.
- Výrobca ponúka za príplatok aj ochranu voči útoku postrannými kanálmi.
- Prevencia voči útokom predlžovaním správy.

4.2.4 Nevýhody

- Na redukovanej verzii sa ukazovali neidálne vlastnosti permutácií.

4.3 JH

Tretím finalistom je návrh hašovacej funkcie, ktorá nesie názov JH. Tento návrh zahŕňa 4 algoritmy - *JH-224, JH-256, JH-384, JH-512*. Tieto algoritmy majú veľmi efektívnu hárdvérovú implemntáciu, pretože sú zložené z

jednoduchých komponentov. Sú veľmi efektívne aj pri softvéroch s SIMD inštrukciami [SI], keď sú komponenty vypočítavané pararelne a to je vhodné pre bit-slice implementáciu.

V dizajne návrhu JH sa využíva nová štruktúra kompresnej funkcie na vytvorenie kompresnej funkcie zo šifry, ktorá využíva široké bloky s konštantným kľúčom. Taktiež sa aplikovala metodológia na "high dimension" , čo umožňuje poskladať široké bloky z jednoduchých komponentov. Vďaka tomuto prístupu dostal tento návrh aj veľmi pozitívne kryptoanalytické hodnotenia.

Pamäťové nároky pri hárdvérovej implementácii sú 1536 bitov (zahŕňajúce už aj 512 bitovú správu). Navyše pri ďalších 256 bitoch je možné generovanie konštánt používaných počas kôl za behu programu. Všetky štyri verzie algoritmu zdieľajú používajú tie isté funkcie, preto je možné ich aplikovať všetky spoločne na hardvér.

Návrh JH má silnú bezpečnostnú stránku. Každý blok správy má 64 bitov a prechádza 42 kolami kopresných funkcií, čo dáva 10752 S-boxov rozmeru $4x4$ bity. Zistilo sa, že diferenciálna cesta pri kompresných funkciách zahŕňa viac ako 700 aktívnych S-boxov, čo spôsobuje vysokú odolnosť voči diferenciálnym útokom.

4.3.1 Elementárna analýza

Štruktúra kompresnej funkcie

Ako sme už spomínali, kompresná funkcia prichádza s novým prítupom, poďme si ho priblížiť.

Kompresná funkcia návrhu JH je skonštruovaná z bijektívnej funkcie (blokovej šifry s konštatným kľúčom). Môžeme si ju pozrieť na nasledujúcom obrázku. Veľkosť bloku šifry je $2m$ bitov. Pre $2m$-bitovú hodnotu hašu $H^{(i-1)}$ a m-bitový blok správy $M^{(i)}$ sa spracujú na haš $H^{(i)}$ o veľkosti $2m$-bitov. Veľkosť vybranej správy je najviac m-bitov.

Táto štruktúra kompresnej funckie je jednoduchá a efektívna. S kľúčom blokovej šifry nastaveným ako konštanta (permutácia), nie sú pridané do stredu funkcie žiadne ďalšie premenné, tak je omnoho jednoduchšie analyzovať bezpečnosť. Bez skrátenia výstupu je táto celkom funkcia efektívna.

Zovšeobecnená metodológia dizajnu AES

AES používa substitučno-permutačnú sieť (SPN) so vstupom dvojrozmerného poľa. "Maximum separable distance"(MDS) kód je vložený do stĺpcov v každom párnom kole (pričom ráta sa od nultého kola) a do riadkov každom nepárnom kole. Kvôli rotácii riadkov sú funkcie počas kôl identické.

Aplikovanie dizajnu AES do vysokých dimenzií umožnil vytvorenie širokého bloku šífry pomocou malých komponentov. V zovšeobecnenej metodológii

dizajnu AES sú vstupné bity rozdelené do $\prod_{i=0}^{d-1} \alpha_i (\alpha_i \geqslant 2)$ častí a tieto časti vytvoria d-dimenzové pole. V lineárnej vrstve r-tého kola, MDS je aplikovaný cez ($rmodd$)- tú dimenziu. Autori veria, že tento spôsob je najefektívnejší na vytvorenie širokého bloku šifry pomocou malých komponentov.

4.3.2 Funkcie

S-boxy

S_0 a S_1 su S-boxy používané v JH o veľkosti 4x4 bity. Namiesto toho aby boli jednoducho XOR-ované so vstupom, v každom kole ukazuje konštantný bit, ktorý S-box sa bude používať pre zvýšenie celkovej algebrickej zložitosti.

x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
S_0	9	0	4	11	13	12	3	15	1	10	2	6	7	5	8	14
S_1	3	12	6	13	5	7	1	9	15	2	0	4	11	10	14	8

Lineárna transfomácia L

Lineárna transformácia L implementuje MDS kód cez $GF(2^4)$. Tu je násobenie $GF(2^4)$ definované ako násobenie binárnych polýnomov modulo ireducibilný polynóm $x^4 + x + 1$. Takéto násobenie označíme ako $\bullet$. Nech A, B, C, D sú 4-bitové slová. L transformácia (A,B) na (C,D) je potom definovaná ako :

$$(C, D) = L(A, B) = (5 \bullet A + 2 \bullet B, 2 \bullet B + A)$$

Permutácia P_d

Permutácia P_d je jednoduchá permutácia 2^d členov. Je to podobné rotovanie riadkov v šifre AES a tak aj používa tie isté funkcie počas kôl pre hardvérovu implementáciu. Skladá sa z funkcii π_d, P'_d a ϕ_d.

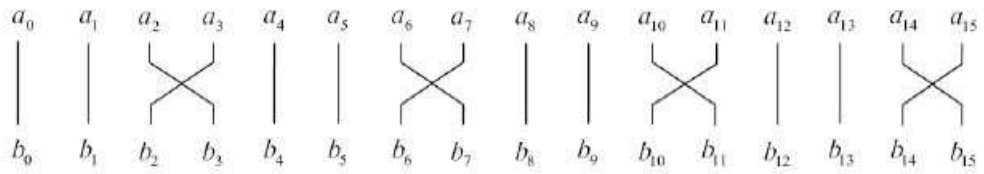
permutácia π_d
π_d operuje s 2^d členmi. Výpočet $B = \pi_d(A)$ je definovaný:

$$b_{4i+0} = a_{4i+0} \text{ for } i = 0 \text{ to } 2^{d-2} - 1;\ b_{4i+1} = a_{4i+1} \text{ for } i = 0 \text{ to } 2^{d-2} - 1;$$
$$b_{4i+2} = a_{4i+3} \text{ for } i = 0 \text{ to } 2^{d-2} - 1;\ b_{4i+3} = a_{4i+2} \text{ for } i = 0 \text{ to } 2^{d-2} - 1;$$

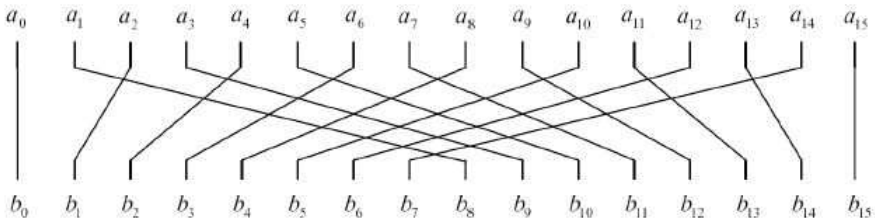
permutácia π_d je zobrazená aj na nasledujúcom obrázku:

permutácia P'_d
P'_d operuje s 2^d členmi. Výpočet $B = P'_d(A)$ je definovaný:

$$b_i{}^{\cdot\cdot} = a_{2i} \text{ for } i = 0 \text{ to } 2^{d-1} - 1;\ b_{i+2^{d-1}} = a_{2i} \text{ for } i = 0 \text{ to } 2^{d-1} - 1;$$



Obrázok 7: Permutácia π_d



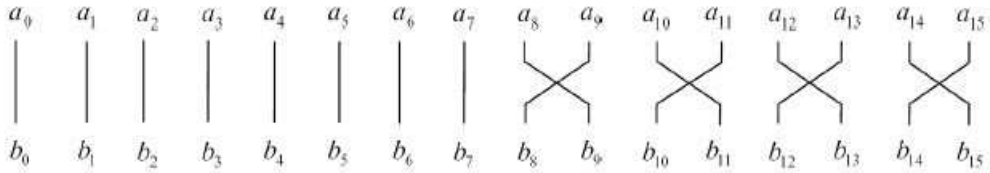
Obrázok 8: Permutácia P'_d

permutácia P'_d je zobrazená aj na nasledujúcom obrázku:

permutácia ϕ_d
ϕ_d operuje s 2^d členmi. Výpočet $B = \phi_d(A)$ je definovaný:

$$b_i\text{¨} = a_i \text{ for } i = 0 \text{ to } 2^{d-1} - 1;\ b_{2i+0} = a_{2i+1} \text{ for } i = 2^{d-2} \text{ to } 2^{d-1} - 1;$$
$$b_{2i+1} = a_{2i+0} \text{ for } i = 2^{d-2} \text{ to } 2^{d-1} - 1;$$

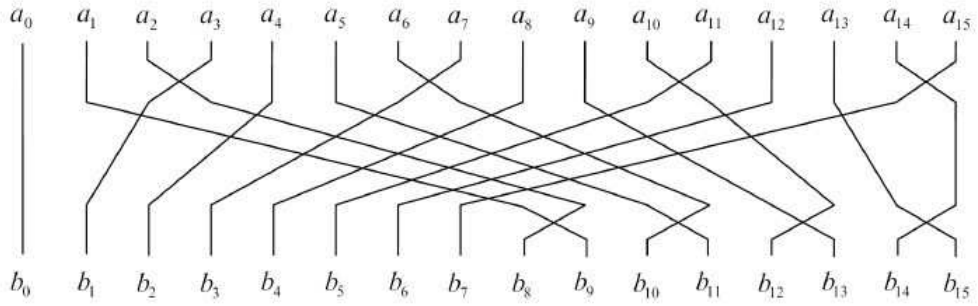
permutácia ϕ_d je zobrazená aj na nasledujúcom obrázku:



Obrázok 9: Permutácia ϕ_d

permutácia P_d
P_d je zložením funkcií π_d, π_d, ϕ_d:

$$P_d = \pi_d \circ \pi_d \circ \phi_d$$



Obrázok 10: Permutácia P_d

permutácia P_d je zobrazená aj na nasledujúcom obrázku:

Funkcia počas kola R_d

Funkcia počas kola implementuje zovšeobecnenú metológiu dizajnu AES, ktorú sme si predstavili pred chvíľou. Obsahuje vrstvu S-boxov, vrstvu lineárnej transformácie a vrstvu funkcie P_d (podobne ako pri AES, kde je to vrstva S-boxov, lineárna transformácia a rotácie riadkov). Vstupy a výstupy funkcie R_d majú 2^{d+2} bitov. Vstupné slovo je označené ako A = $(a_0 \parallel a_1 \parallel ... \parallel a_{2^{d-1}})$, kde každé a_i reprezentuje štvorbitové slovo. Výstupné slovo je označené ako B = $(b_0 \parallel b_1 \parallel ... \parallel b_{2^{d-1}})$, kde každé a_i reprezentuje štvorbitové slovo. 2^d-bitové konštanty pre každé kolo je v r-tom kole označená ako $C_r^{(d)} = C_r^{(d),0} \parallel C_r^{(d),1} \parallel ... \parallel C_r^{(d),2^d-1}$. Výpočet $B = R_d(A, C_r^{(d)})$ je daný nasledovne:

1. for $i = 0$ to $2^d - 1$
   {
   if $C_r^{(d),i} = 0$ then $v_i = S_0(a_i)$;
   if $C_r^{(d),i} = 1$ then $v_i = S_1(a_i)$;
   };
2. $(w_{2i}, w_{2i+1}) = L(v_{2i}, v_{2i+1})$ pre $0 \leqslant i \leqslant 2^{d-1} - 1$;
3. $(b_0, b_1, ..., b_{2^d-1}) = P_d(w_0, w_1, ..., w_{2^d-1})$;

4.3.3 Bezpečnosť

Výrobca udáva, že je potrebných najmenej $2_{m/2}$ operácií na nájdenie kolízie. A na nájdenie vzoru je potrebných 2^m operácií. Ďalšie útoky, ktoré boli skúmané výrobcami nebudeme rozoberať, keďže sme sa týmto útokom nikde nevenovali. V prípade záujmu sa o nich môžeme dočítať na stránke výrobcu.

4.3.4 Výhody

JH má nasledovné výhody:

- jednoduchý dizajn
- štruktúra kompresnej funkcie dáva možnosť jednoduchého a efektného výpočtu
- zovšeobecnený návrh šifry AES dáva možnosť skonštruovať veľký blok šifry z jednoduchých komponentov
- bezpečnostné nedostatky môžu byť jednoducho odstránené
- vysoká efektívnosť pri obrane voči kolíziám
- JH môže byť efektívne implementovaný na hardveri a 128/256 - bitových procesoroch.
- je možné využiť JH na nahradenie väčšiny aplikácii, ktoré využívajú SHA-2.

4.4 Keccak

Keccak je rodinou hašovacích funkcií postavených na konštrukcii typu Sponge, ktorá na zostrojenie bloku permutácie využíva 7 rôznych permutácií.

4.4.1 Elementárna analýza

Funkcie $Keccak - f$

Existuje 7 rôznych funkcií $Keccak - f$, indikované cez $Keccak - f[b]$, kde $b = 25x2^l$ a l je z hodnôt od 0 po 6. $Keccak - f[b]$ je permutácia nad poľom Z_2^b, kde sú bity označované s a indexované od 0 po $b - 1$. Hodnotu b voláme aj šírka permutácie.

Permutácia $Keccak - f[b]$ je opísaná ako postupnosť operácií na stave a, ktoré je definovaná ako 3D pole, zložené z prvkov $GF(2)$, menovite $a[5][5][w]$, kde $w = 2^l$. Vyjadrenie $a[x][y][z]$ pričom $x, y \in Z_3$ a $z \in Z_w$, opisuje bitovú pozíciu x, y, z. Indexovanie začína nulou. Mapovanie medzi pozíciou s a prislúchajúceho a sa vypočíta ako: $s[w(5y + x) + z] = a[x][y][z]$. Počítanie hodnôt x a y je počítané modulo 5, hodnota z sa počíta modulo w. Niekedy sa môže vynechávať index [z], ale obidva zápisy indikujú, že údaj je správny pre všetky hodnoty vynechaného indexu.

$Keccak-f[b]$ je iterovanou permutáciou, pozostávajúcou s z postupnosti n_r kôl R, indexovaných i_r pre hodnoty od 0 po $n-1$. Jedno kolo pozostáva z piatich častí:

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

$\theta : a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1]$,
$\rho : a[x][y][z] \leftarrow a[x][y][z-(t+1)(t+2)/2]$,
kde t je $0 \leqslant t \leqslant 24$ a $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$ v $GF(5)^{2x2}$,
alebo $t = -1$ ak $x = y = 0$
$\pi : a[x][y] \leftarrow a[x'][y']$, keď $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$,
$\chi : a[x] \leftarrow a[x] + (a[x+1]+1)a[x+2]$,
$\iota : a \leftarrow a + RC[i_r]$,

Sčítavanie a násobenie medzi jednotlivými časťami sa odohráva v poli GF(2). S výnimkou konštanty pre kolo (RC), sú jednotlivé kolá totožné. Konštanta pre kolo sa vypočíta ako:

$$RC[i_r][0][0][2^j - 1] = rc[j + 7i_r] \text{ pre všetky hodnoty } 0 \leqslant j \leqslant l$$

a všetky ostatné hodnoty $RC[i_r][x][y][z]$ sú nastavené na hodnotu nula. Hodnoty $rc[t] \in GF(2)$ sú definované ako výstup binárneho spätného posuvného registra (LFSR):

$$rc[t] = (x^t mod x^8 + x^6 + x^5 + x^4 + 1) mod x \text{ v GF(2)[x].}$$

Počet kôl n_r je určený pomocou šírky permutácie, konkrétne

$$n_r = 12 + 2l.$$

Sponge konštrukcia

Sponge konštrukcia[SP1] je založená na funkcii $SPONGE[f, pad, r]$, ktorá spracováva vstup rozdielnej dĺžky na výstup ľubovoľnej dĺžky za pomoci permutácie f pre fixnú dĺžku, pravidla na dopĺňanie "pad" a parametra *bitrater*. Permutácia f operuje na fixnom počte bitov, označeného ako *widthb*. Hodnota $c = r - b$ sa nazýva kapacita.

Pre pravidlo dopĺňania správy sa používa nasledujúce označenie: dopĺňanie správy M do sequencie po x-bitových blokoch je označené ako $M \| pad[x](|M|)$, kde $|M|$ je dĺžka správy M v bitoch.

Na začiatku má stav hodnotu 0^b, ktorý sa nazýva aj rootovací stav. Rootovací stav má pevnú hodnotu a nikdy by nemal byť použitý ako vstup. Je to podľa výrobcov kľučové pre bezpečnosť návrhu.

Algoritmus potom funguje nasledovne:
Požadované:$r < b$
Rozhranie: $Z = sponge(M, l)$ keď $M \in Z_2^*$, integer $l > 0$ a $Z \in Z_2^l$
$P = M \| pad[r](|M|)$ $s = 0^b$
for $i = 0$ to $|P|_r - 1$ **do** $s = s \oplus (P_i \| 0^{b-r})$; $s = f(s)$;
endfor $Z = \lfloor s \rfloor_r$
while $|Z|_r r < l$ **do** $s = f(s)$; $Z = Z \| \lfloor s \rfloor_r$;
endwhile return $\lfloor Z \rfloor_l$

4.4.2 Bezpečnostné požiadavky kladené na Keccak

Pre každú podporovanú hodnotu parametra výrobcovia vytvorili "flat sponge claim". o ktorom sa dočítate viac v [SP1]. Výrobcovia očakávajú úspešnú pravdepodobnosť proti akémukoľvek útoku na funkciu $Keccak[r, c]$ s náplnením N volaní funkcie $Keccak - f[r + c]$ alebo jeho opak by mal byť menší alebo rovný ako pre náhodné orákulum:

$$1 - \exp(-N(N + 1)2^{-(c+1)}).$$

4.4.3 Výhody

- **Jednoduchosť:** V porovnaní s ostatnými konštrukciami, ktorých horná hranica bola úspešná voči všeobecným útokom , je konštrukcia sponge veľmi jednoduchá a navyše sa táto hranica dá jednoducho zvýšiť.
- **Voliteľná dĺžka výstupu:** Dokáže generovať ľubovoľnú dĺžku výstupu a jedna funkcia môže byť použitá pre rôzne dĺžky.
- **Flexibilita:** Bezpečnostná úroveň môže byť zvýšená na úkor rýchlosti, pri použití rovnakej permutácie.

4.5 Skein

Posledným finalistom suťaže je návrh Skein. Jeho autory sa o ňom vyjadrili nasledovne: "Skein je novou rodinou hašovacích funkcií. Je rýchly, bezpečný, jednoduchý a flexiblný v štandardnom balení a teda sa dá jednoducho skúmať. " Poďme sa teda pozrieť,či autori môžu toto svoje tvrdenie podložiť aj faktami. Navyše sa o ňom mnoho odborníkov vyjadruje ako o najväčšom favoritovi[CR2].

Skein je rýchly. Dosahuje výborné výkonostné výsledky na mnohých platformách.

Skein je bezpečný. Návrh je založený na konzervatívnom prístupe postavenom na blokovej šifre Threefish, ktorá ukazuje omnoho lepšie bezpečnostné vlastnosti ako napríklad šifra AES. Navyše Skein má mnoho vlastnosti, ktoré boli vložené do algoritmu, aby ešte zvýšili jeho bezpečnosť.

Skein je jednoduchý. Používa iba tri jednoduché operácie, preto kompresná funkcia je jednoducho pochopiteľná a zapamätateľná. Zvyšok algoritmu je priame iterovanie tejto funkcie.

Skein je flexibilný. Skein je definovaný pre 3 veľkosti vnútorného stavu: 256, 512 a 1024 bitov a ľubovoľnú veľkosť výstupu. To umožňuje návrhu Skein byť vhodným nástupcom doterajších SHA hašovacích funkcií. Navyše návrh ako taký umožňuje využitie Skein aj v iných oblastiach ako hašovanie (napr. ako prúdovú šifru, či ako funkcia na odvodenie kľúča)

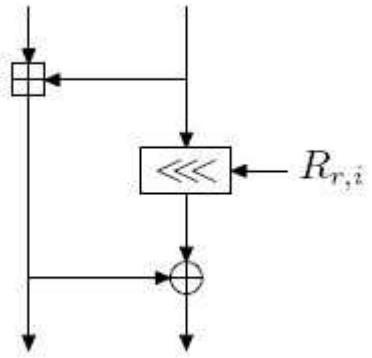
4.5.1 Elementárna analýza

Ako sme už spomínali, Skein je rodinou hašovacích funkcií s 256,512 a 1024 bitovými vnútornými stavmi, ktoré dokážu vytvoriť výstup akejkoľvek veľkosti. My sa budeme venovať Skein-512, pretože táto verzia je základom návrhu. Inovatívnou myšlienkou návrhu Skein je postaviť na rotačnej blokovej šifre. Použitie takejto šifry umožňuje konfigurovať hašovacie dáta na základe vstupného textu v každom bloku a v každom prípade urobiť kompresnú funkciu jedinečnou. Táto vlastnosť ukazuje mnoho útokov na hašovacie funkcie a vynikajuco zlepšuje flexibilitu návrhu Skein.
Konkrétne, Skein je postavený na týcho troch nových komponentoch:

- **Threefish** Threefish je rotačná bloková šifra v jadre návrhu Skein, definovaná pre veľkost bloku 256,512 a 1024 bitov.
- **Unique Block Iteration (UBI)**UBI, teda "jedinečná bloková iterácia" je zreťazujúci mód, ktorý používa Threefish na vytvorenie kompresnej funkcie tak, že spracováva ľuboľnú veľkosť vstupu na pevnú dĺžku výstupu.
- **Systém voliteľných argumentov** Táto vlastnosť umožňuje podporovať rôzne nastaviteľné vlastnosti bez veľkých prídavkov pri implementácií a aplikácií, ktoré konkrétne nastavenie nepoužíva.

Bloková šifra Threefish Threefish je široká, rotačná bloková šifra. Je definovaná pre 256, 512 a 1024 bitov. Veľkosť kľuča je rovnaká ako veľkosť bloku a rotačná hodnota je 128 bitov pre všetky typy blokov.

Jadrom návrhu šifry Threefish je to, že veľký počet jednoduchých kôl je bezpečnejší ako nižší počet komplexnejších kôl. Threefish využíva len tri matematické operácie: XOR, sčítavanie, a konštantné rotácie - na 64 bitových slovách - čo je obzvlášť efektívne na moderných 64 bitových procesoroch. Tieto tri operácie sa používajú vo funkcii MIX, ktorá je základom šifry.



Obrázok 11: funkcia MIX

V každom zo 72 kôl Skein-512 pozostáva zo štyroch funkcii MIX nasledovaných osmimi 64-bitovými slovami. Každé 4 kolá je potom ešte vkladaný subkľúč. Permutácia slov je rovnaká pre všetky kolá a rotačná konštanta je vyberaná pre maximálnu difúziu a opakuje sa každých 8 kôl.

Tabuľka kľúčov generuje subkľúče z kľúča a rotácie. Každý subkľúč pozostáva z troch častí: kľučových slov, rotačných slov a kontrolného súčtu. Na vytvorenie tabuľky kľúčov sú kľúč a rotácia rozšírené o jednu extra paritné slovo, ktoré je XOR-om všetkých ostatných slov. Celá tabuľka môže byť vytvorená iba v pár procesorových cykloch.

UBI

Zreťazujúci mód UBI kombinuje vstupnú zreťazenú hodnotu s ľubovoľne dlhým vstupným reťazcom a produkuje výstup pevnej dĺžky.

Srdcom UBI je rotácia. Pri použití rotačnej šifry je jasné, že každý blok je spracovávaný jedinečnou verziou kompresnej funkcie. Toto ukončuje množstvo útokov "cut&paste", pretože časť správy je spracovaná na jeden výsledok ale v inej lokácii je spracovaná na úplne iný vysledok.

Hašovanie pomocou Skein

Skein je založený na viacnásobnom vyvolaní módu UBI. Začína so zreťazenou hodnotou 0 a následne je 3x zavolaný mód UBI, prvýkrát pre konfiguračný

blok, druhýkrat na spracovanie správy (až do dĺžky $2^{96} - 1$ bitov)a tretíkrát výstupnú transformáciu

Konfiguračný blok má veľkost 32 bajtov a je tam uvedená dĺžka výstupu a ďalšie parametre, ktoré ale pre našu analýzu nie sú podstatné.

Výstupná transformácia je schopná dosiahnuť vhodnú úroveň znáhodnenia pre hašovanie. Taktiež umožňuje vytvárať akúkoľvek veľkosť výstupu až po 2^{64} bitov. Ak jeden výstupný blok nie je postačujúci, opakuje sa niekoľkokrát. Používanie veľkých výstupov je samozrejme vhodné, ale úroveň bezpečnosti záleží od veľkosti vnútorného stavu.

Nastaviteľné argumenty Spomínali sme, že Skein je veľmi flexibilný, tu je zoznam argumentov, ktoré je možné nastaviť:

- **Kľúč**(voliteľný): Je potrebný pre splnenie bezpečnosti
- **Konfigurácia**(povinný): Konfiguračný blok spomínaný kúsok vyššie.
- **Personalizácia**(voliteľný): Umožňuje aplikácii vytvoriť rozdielne funkcie pre rozdielnych užívateľov.
- **Verejný kľúč**(voliteľný)
- **Identifikácia odvodeného kľúča**(voliteľný)
- **Nonce**(voliteľný): Hodnota používaná pri prúdových šifrách.
- **Správa**(voliteľný): Správa ako vstup do hašovacej funkcie.
- **Výstup**(povinný): Výstupna transformucia.

4.5.2 Bezpečnosť

Základné bezpečnostné požiadavky

Skein bol navrhnutý tak, aby bol bezpečný pre široké spektrum aplikácií, zahrňujúc digitálny podpis, vytváranie kľúčov, generovanie psuedonáhodných čísel a použitie ako prúdová šifra.

Hodnota n znamená veľkosť vnútorného stavu a m najmenšiu možnú veľkosť vnútorného stavu a výstupu. Hodnoty vyjadrujú úroveň bezpečnosti pri použití štandardných útokov(Birthday attack, Joux):

- Nájdenie prvého vzoru: 2^m
- Nájdenie druhého vzoru: 2^m

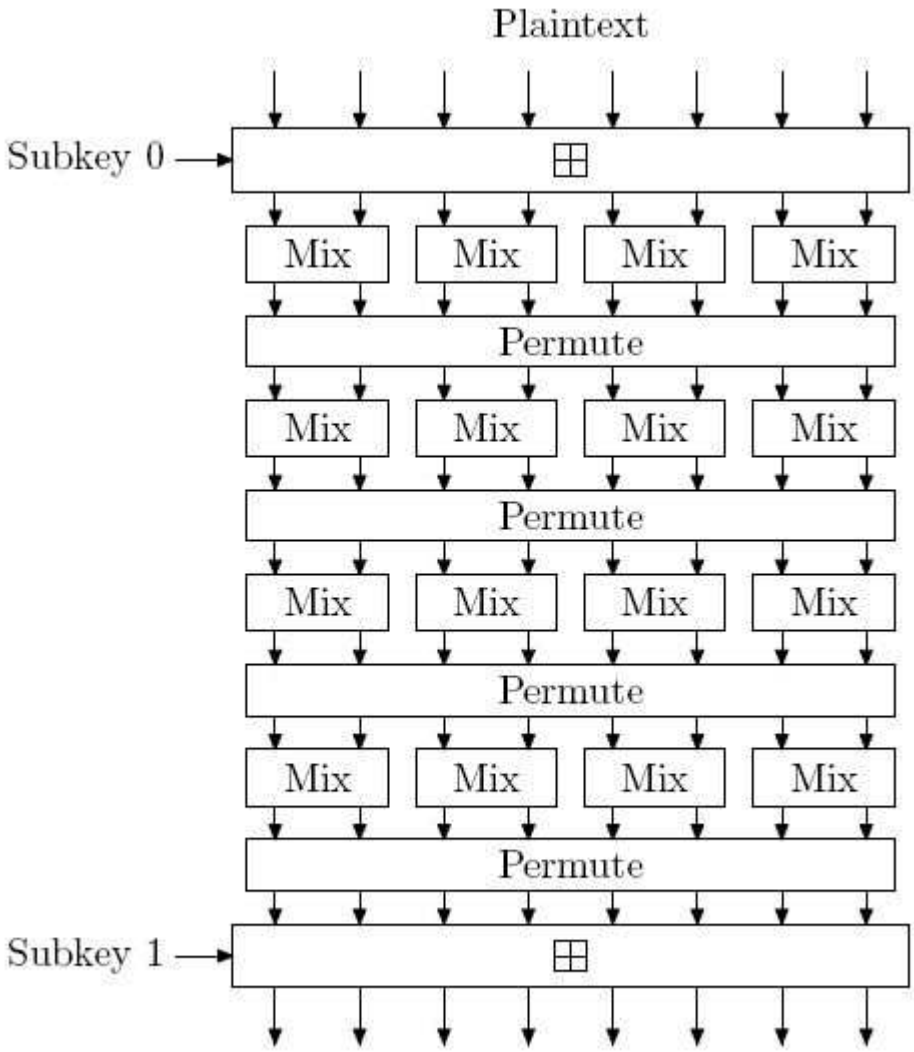
- Nájdenie kolízie: $2^{m/2}$
- Nájdenie k-tej kolízie hrubou silou $\min 2^{n/2}, 2^{r-1}m/k$.(k-tá kolízia pozostáva z k rozličných správ $M_1, ..., M_k s H(M_1) = H(M_2) = ... = H(M_k)$)

Mód UBI v návrhu Skein je odolný aj voči útoku predlžovaním správy: Ak entrópia správy M je dostatočne veľká, tak, že útočník nedokáže uhádnuť M, tak pravdepodobnosť úspechu takéhoto útoku je 2^{-m}.

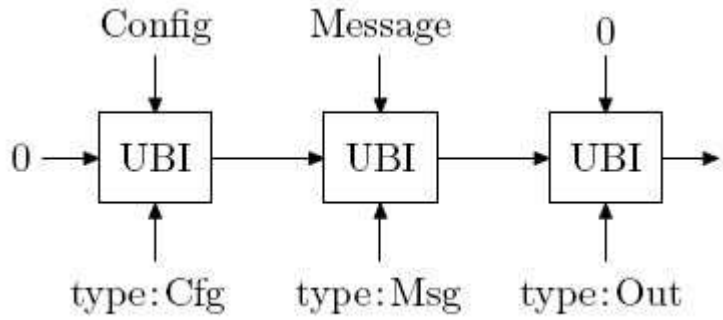
Útoky sa podarili na redukované verzie. Pre 24-26 kôl. Keďže ale návrh používa 72 a 80 kôl v plnej verzii, sú takéto výsledky úspechom výrobcov.

4.5.3 Výhody

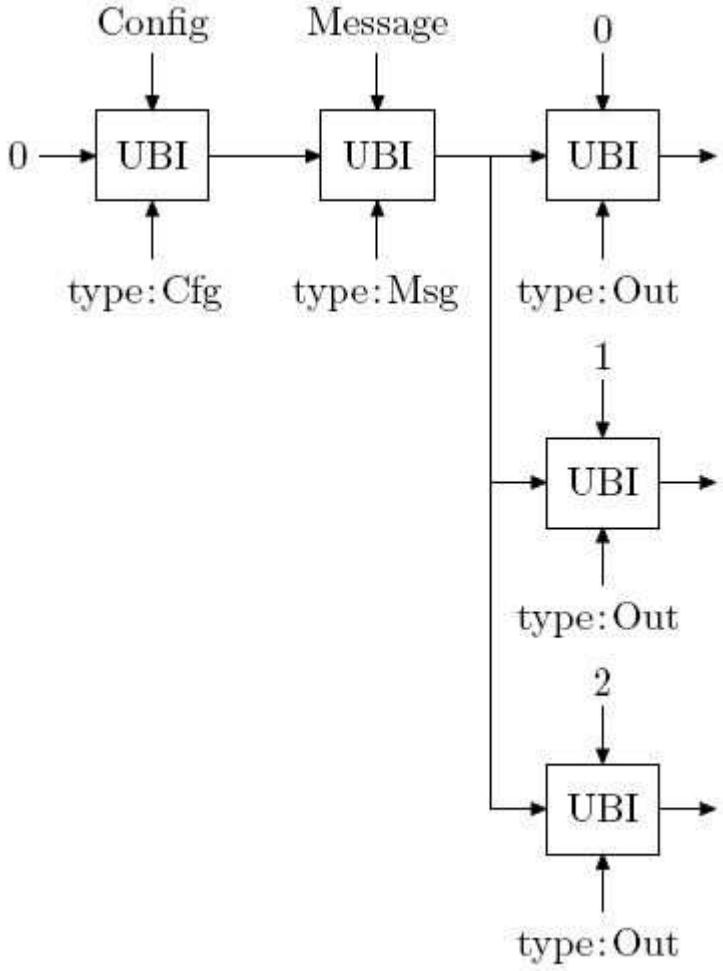
- **Jednoduchosť dizajnu**: dizajn je založený na jednoduchej rotačnej šifre Threefish
- **Rýchlosť**: Keďže návrh využíva len jednoduché operácie má vysoký výkon
- **Bezpečnosť**: Výborná výpočtová zložitosť pri útokoch.



Obrázok 12: štyri zo 72 kôl blokovej šifry Threefish-512



Obrázok 13: hašovanie pomocou Skein



Obrázok 14: hašovanie s viacerými výstupnými blokmi

5 Experiment

V poslednej časti tejto bakalárskej práce sa budeme venovať malému experimentu. Na stránke NIST[NI] boli zverejnené zdrojové kódy všetkých kandidátov tretieho kola. Predmetom tohto experimentu bude odmerať a porovnať rýchlosť hašovania jednotlivých návrhov.

5.1 Použité nástroje

Pri prevádzaní experimentu sme použili nasledovné nástroje:

- Hardvér: Procesor Intel Core DUO 2.0GHz, 3GB RAM
- Operačný systém: Windows XP Professional
- Rozharanie: DEV-C++ v.4.9.2.2

5.2 Priebeh

Po implementácii zdrojového kódu na našom počítači, sme vytvorili vstupnú správu, ktorou je pole o veľkosti 1000000 znakov, pre hašovaciu funkciu a následne sme 1000000x opakovali hašovanie. Výsledný čas sme zapísali do tabuľky. Pokus sme opakovali 5x pre každú verziu jednotlivých kandidátov.

5.3 Tabuľka

kandidát	1.čas	2.čas	3.čas	4.čas	5.čas	priemer
BLAKE-224	4 782	5 078	4 766	5 046	6 157	5 165,8
BLAKE-256	4 078	4 343	3 796	4 484	4 531	4 246,4
BLAKE-384	8 734	9 203	7 860	9 532	9 938	8 712,4
BLAKE-512	8 093	8 454	7 234	8 718	8 859	8 271,6
Grøstl-224	22 610	19 641	22 860	21 860	20 485	21 491,2
Grøstl-256	23 438	23 500	22 093	22 812	22 640	22 896,6
Grøstl-384	55 625	59 109	56 313	56 000	42 297	53 868,8
Grøstl-512	59 453	61 141	53 875	58 313	51 656	56 887,6
JH-224	99 843	98 031	101 439	91 625	92 735	96 734
JH-256	99 125	103 907	102 531	94 219	92 015	98 359,4
JH-384	138 328	143 656	153 969	147 078	151 313	146 868,0
JH-512	147 032	152 031	153 391	148 397	144 562	149 062,6
Keccak-224	9 609	10 758	12 891	8 969	11 125	10 670,4
Keccak-256	8 969	9 141	11 172	9 328	11 266	9 975,2
Keccak-384	8 329	8 265	11 797	8 781	10 437	9 521,8
Skein-256	11 218	11 516	11 938	11 266	11 890	11 565,6
Skein-512	12 172	12 437	12 922	12 359	11 782	12 334,4
Skein-1024	19 578	20 984	19 328	21 984	21 219	20 618

**poznamka: všetky časy v tabuľke sú uvedené v ms.*

5.4 Zhodnotenie výsledkov experimentu

Ako je jednoznačne v tabuľke vidieť najrýchlejším návrhom je BLAKE, nasledovaný návrhmi Keccak a Skein. Najpomalším návrhom je podľa nášho experimentu JH. Tieto naše výsledky korešpondujú z názorom [CR2], že najväčšími favoritmi na víťazstvo sú BLAKE a Skein. Samozrejme rýchlosť nie je jediný aspekt, ktorý sa v suťaži skúma, ale minimálne podľa rýchlosti sa môžeme s týmto názorom stotožniť.

Použité zdrojové kódy sú aj súčasťou prílohy.

Záver

V tejto práci sme sa venovali hašovacím funkciám. V prvej časti ich vlastnostiam, konštrukciám a bezpečnosti, čo obsahovalo naštudovanie si danej problematiky omnoho podrobnejšie ako je v učebných osnovách. V druhej časti sme potom opisovali jednotlivých kandidátov, ktorí postúpili do tretieho a zároveň posledného kola súťaže NIST o nový štandard v oblasti hašovacích funkcií. Snahou bolo, aby prehľad, ktorý vytvoríme bol prehľadný a zrozumiteľný, v rámci možností sa nám to snáď aj podarilo. Pevne veríme že táto bakalárska práca poslúži ľuďom, ktorí budú potrebovať informácie o jednotlivých návrhoch zo súťaže NIST.

Príloha

Priložené CD obsahuje:

- elektronickú podobu práce
- zdrojové kódy kandidátov originál
- zdrojové kódy kandidátov upravené pre účel merania rýchlosti hašovania

Referencie

[MI1] Mgr. Michal Mikuš: *Kryptograficky silné hašovacie funkcie, diplomová práca, 2007.*

[Cha1] Daniel J. Bernstein: *ChaCha, a variant of Salsa20. cr.yp.to/chacha.html.*

[LA1] Jean-Philippe Aumasson, Willi Meier, and Raphael C.-W. Phan: *The hash function family LAKE. In FSE, 2008.*

[RI1] J. Daemen and V. Rijmen: *AES Proposal: Rijndael. AES Algorithm Submission, September 1999, dostupné na http://csrc.nist.gov/archive/aes/rijndael/ Rijndael-ammended.pdf.*

[RI2] J. Daemen and V. Rijmen: *The Design of Rijndael. Springer, 2002.*

[RA1] Martin Boesgaard, Mette Vesterager, Thomas Pedersen, Jesper Christiansen, and Ove Scavenius: *Rabbit: A new high-performance stream cipher. FSE, 2003.*

[SP1] *Cryptographic sponge functions, January 2011. http://sponge.noekeon.org/.*

[CR2] *Crypto-World,číslo12/2010. Mgr. Pavel Vondruška: Finále SHA-3 – jak to vidím já, http://crypto-world.info/casop12/crypto12_10.pdf.*

[FO08] P.-A. Fouque, J. Stern, and S. Zimmer: *Cryptanalysis of Tweaked Versions of SMASH and Reparation. In R. Avanzi, L. Keliher, and F. Sica, editors, Selected Areas in Cryptography 2008, Proceedings, volume 5381 of Lecture Notes in Computer Science, pages 136-150. Springer, 2009.*

[NI] *http://csrc.nist.gov/groups/ST/hash/sha-3/index.html*

[BL] *http://www.131002.net/blake/*

[GR] *http://www.groestl.info/*

[JH] *http://www3.ntu.edu.sg/home/wuhj/research/jh/*

[KE] *http://keccak.noekeon.org/*

[SH] *http://www.skein-hash.info/*

[HA] $http://csrc.nist.gov/groups/ST/hash/documents/DUNKELMAN_NIST3.pdf$

[SP] *http://sponge.noekeon.org/SpongeFunctions.pdf*

[MOV97] A.J.Menezes, P.C.Van Oorschot and S.A.Vanstone:: *CHandbook of Applied Cryptography, chapter Hash Functions and Data Integrity, p. 321–383. CRC Press, 1997.*

[JO04] A.Joux:: *Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. Advances in Cryptology-CRYPTO 2004, volume 3152 of Lecture Notes in Computer Science, p. 306–316, Springer, 2004.*

[DA01] Joan Daemen and Vincent Rijmen:: *The Wide Trail Design Strategy, Springer,2001.*

[DA02] Joan Daemen and Vincent Rijmen:: *TAES and the Wide Trail Design Strategy, Springer, 2002.*

[SI] *ENCYCLOPEDIA OF MULTIMEDIA, Springer 2008, Part 19, 817-819.*

[CR10] *Cryptographic Hash Functions, Springer 2010, 2010, Part A, 59-79.*

[ZK10] Otokar Grošek, Milan Vojvoda, Marcel Zanechal, Pavol Zajac:: *Základy kryptografie, STU,2010.*