

Kryptografické hašovací funkce - MAC, MDC kódy

často stačí pouze autentizace původu či obsahu zprávy, případně ověření integrity
ne vždy stačí k zajištění těchto požadavků vlastní šifrovací algoritmus

Použití

- symetrické systémy - slouží k rozpoznání pravosti dešifrované zprávy
- asymetrické systémy - rychlejší autentizace: spočítá se hašovací funkce nad danou zprávou a elektronicky se podepíše až výsledný hashkód
- ochrana hesel a passphrases

Závisí-li výpočet hašovací funkce na tajném klíči, označujeme tuto funkci jako MAC (message authentication code). Pokud takový klíč použit není, jde o MDC (manipulation detection code).

Semi-formální definice

MAC je funkce h splňující

1. zatímco argument X může být libovolné délky, výsledek $h(K, X)$ má pevnou velikost n ($n \geq 32 \dots 64$) 128 ..256
2. pro dané h a X je těžké určit $h(K, X)$ s pravděpodobností úspěchu výrazněji převyšující $1/2^n$.
3. i se znalostí velkého množství zvolených párů $\{X_i, h(K, X_i)\}$ musí být těžké určit K , nebo spočítat $h(K, X)$ pro $X \neq X_i$.

MDC často dělíme na dvě skupiny - OWHF, CRHF

Funkci h označíme za OWHF (one-way hash function) právě když:

1. zatímco argument X může být libovolné délky, výsledek $h(X)$ má pevnou velikost n ($n \geq 128$, dnes spíše 160 ... 256, 512 a více)
2. pro dané Y a h musí být těžké nalézt X aby $Y = h(X)$ (tzv. *preimage resistance*)
3. pro dané X a $h(X)$ musí být těžké najít X' aby $h(X) = h(X')$ (tzv. *second preimage resistance*)

CRHF (collision resistant hash function) je taková funkce, která splňuje:

1. h je OWHF ($n \geq 128$, dnes spíše 160 a více)
2. musí být těžké najít pár X', X'' , tak, aby $h(X') = h(X'')$ (tzv. *collision resistance*)

není známa žádná dokazatelně bezpečná hašovací funkce, pouze známe funkce ekvivalentní s NP-úplnými problémy

Návrhy MDC funkcí

prakticky všechny známé hašovací funkce pracují nad vstupem pevné délky
iterované hašovací funkce (iterated hash f.) - delší vstup X je zpracováván opakováním výpočtu

$$\begin{aligned} H_0 &= IV \\ H_i &= \mathbf{f}(X_i, H_{i-1}) \quad i = 1, 2, \dots, t \\ \mathbf{h}(X) &= H_t \end{aligned}$$

IV - inicializační hodnota (Merkle-Damgard konstrukce)

často délka vstupu není násobkem délky vstupního bloku $\rightarrow$ nutné zarovnání (padding)

zarovnání by mělo být jednoznačné (unambiguous) - nesmí dvě různé zprávy doplnit na stejnou, je vhodné, aby na konec zprávy zapisovalo délku

Tvrzení

Pokud zarovnání obsahuje délku vstupní zprávy a tato je dlouhá alespoň 2 bloky, potom nalezení kolidujícího vzoru $\mathbf{h}$ při pevném IV vyžaduje 2^n operací právě když nalezení kolidujícího vzoru $\mathbf{f}$ při libovolném H_{i-1} vyžaduje 2^n operací.

Tvrzení

Předpokládejme jednoznačné zarovnání obsahující délku vstupní zprávy. Je-li $\mathbf{f}$ odolná vůči kolizím (collision resistant), je $\mathbf{h}$ CRHF.

Hašovací funkce založené na blokových šifrách

výhodné z hlediska návrhu a implementace

většina schémat ekvivalentní s jedním z následujících

Matyas, Meyer, Oseas

$$\mathbf{f} = E^{\oplus}(s(H_{i-1}), X_i)$$

délka hashkódu odpovídá délce vstupu, $E^{\oplus}(K, X) = E(K, X) \oplus X$.

ISO/IEC 10118 p. 2

Preneel, Govaerts, Vandenwalle

$$\mathbf{f} = E^{\oplus}(s(H_{i-1}), X_i) \oplus H_{i-1}$$

v obou případech E je libovolná bloková šifra, s je funkce mapující prostor zašifrovaných bloků do prostoru klíčů. Tyto funkce mohou být CRHF pokud délka bloku je alespoň 128 bitů. Bezpečnost úměrná hodnotě $\min(k, r)$.

MDC-2

funkce dává hashkód dvojnásobné délky vstupu, vhodná pro implementaci CRHF pomocí DES.... Dnes cokoliv moderního

Občas pod názvem Meyer-Schilling

$$T1_i = E^{\oplus}(H1_{i-1}, X_i) = LT1_i \parallel RT1_i$$

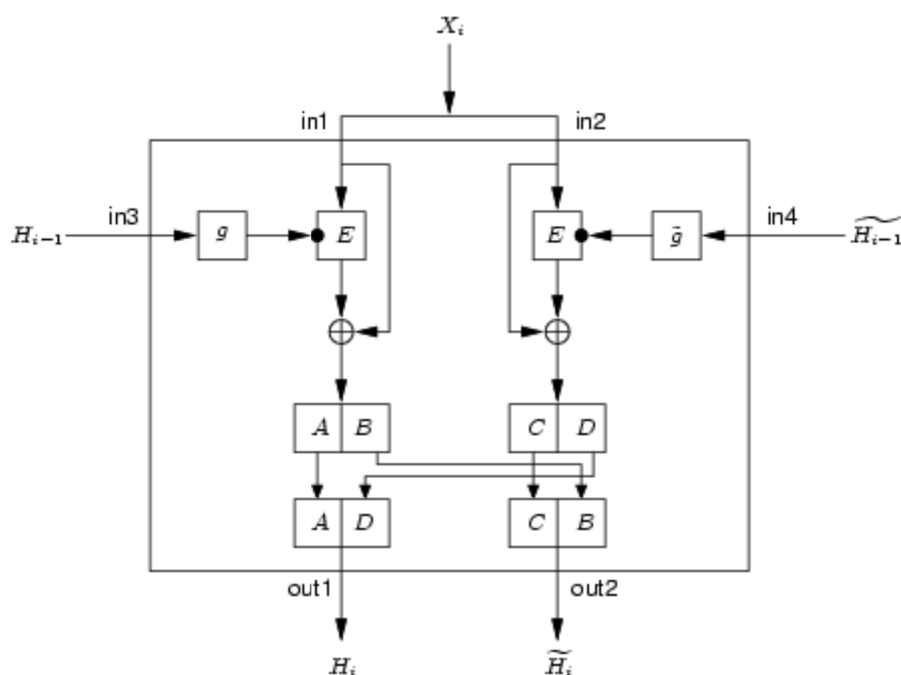
$$T2_i = E^{\oplus}(H2_{i-1}, X_i) = LT2_i \parallel RT2_i$$

$$H1_i = LT1_i \parallel RT2_i$$

$$H2_i = LT2_i \parallel RT1_i$$

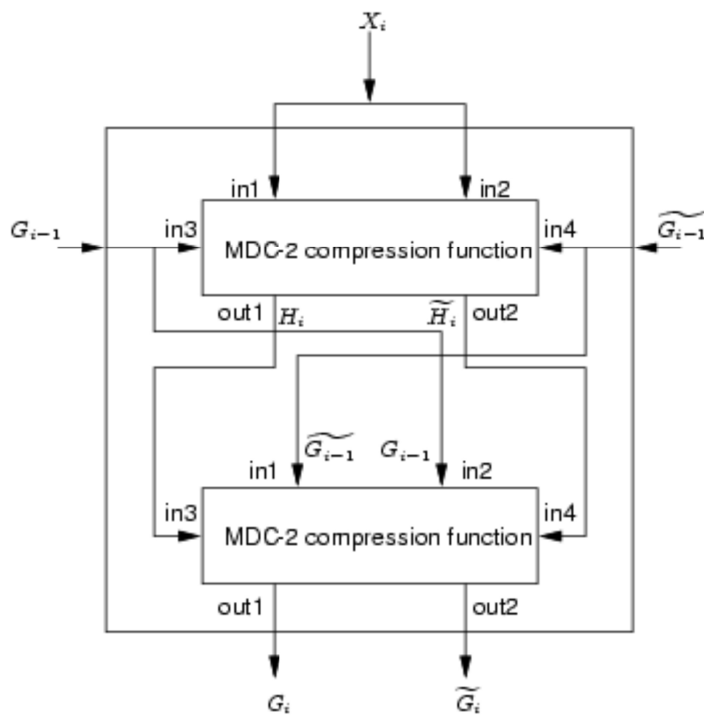
$H1_0$ a $H2_0$ iniciovány hodnotami IV_1 resp. IV_2 , výsledek vznikne konkatenací $H1_i$ a $H2_i$.

ještě obrázek:



MDC-4

jeden krok se skládá ze dvou iterací MDC-2, jako vstup druhé iterace se použije $H2_{i-1}$ a $H1_{i-1}$.



Funkci dnes považujeme za bezpečnou při použití bezpečného šifrovacího algoritmu.

Hašovací funkce založené na modulární aritmetice

nejlepší schémata založena na umocňování na druhou modulo n

$$f = (X_i \oplus H_{i-1})^2 \bmod N \oplus X_i$$

použitím většího množství operací lze ještě zvýšit bezpečnost

$$f = (H_{i-1} \oplus (X_i)^2)^2 \bmod N$$

Hašovací funkce založené na problému batohu

zatím není jasné, zda problém batohu je těžký pouze v nejhorším případě, nebo zda je těžký v průměrném případě

prakticky všechny systémy založené na problému batohu byly prolomeny

Speciální MDC funkce

algoritmy navrhované od počátku jako výpočet hašovacích funkcí a potažmo MDC obvykle výkonnější

MD4, MD5

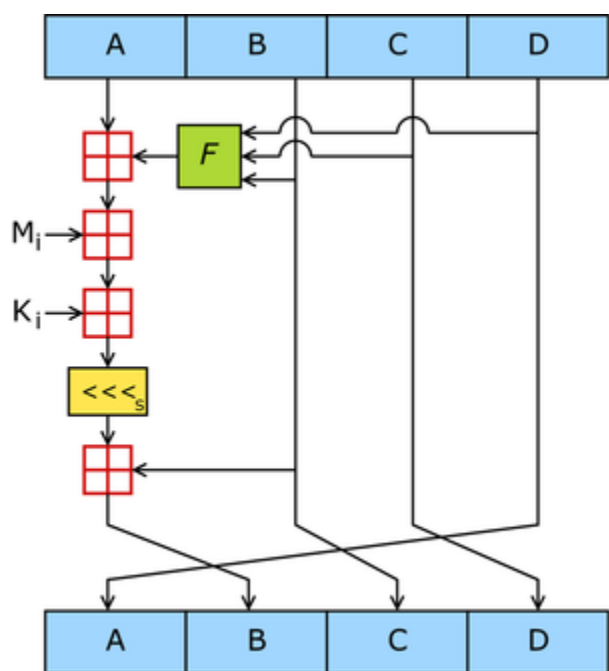
MD4 navržen Rivestem, produkuje 128 bitový výstup, iterace ve třech cyklech

publikován útok proti prvním dvěma cyklům algoritmu
 autor provedl vylepšení → MD5

MD5 pracuje nad vstupními bloky délky 512 bitů, 128 bitový výstup, pracuje ve čtyřech cyklech

algoritmus patří do stejné „rodiny“ hashovacích funkcí, jako SHA1 a má i obdobnou strukturu

dle posledních výsledků lze oba algoritmy považovat za prolomené, což platí obecně pro všechny algoritmy s délkou „digestu“ do 160 bitů



SHA-1

hašovací funkce transformující zprávu libovolné délky na 160-bitový digest. Pracuje ve třech krocích:

- inicializace – zarovnání vstupu, příprava interních datových struktur
- iterace kompresní funkce – iterativní aplikace kompresní funkce na bloky zprávy, akumulace historie výpočtu
- dokončení – konstrukce finálního výsledku z akumulovaných vnitřních stavů

Inicializace

nejprve se doplní zpráva do násobku 512: doplní se 1, potom tolik nul, aby zpráva byla o 64 kratší než nejbližší vyšší násobek 512 a na konec do 64 bitů zapsaná původní délka zprávy

Inicializace řetězcích proměnných:

$$H_1 = 0x67452301$$

$$H_2 = 0xefcdab89$$

$$H_3 = 0x98bacdf$$

$$H_4 = 0x10325476$$

$$H_5 = 0xc3d2e1f0$$

Iterace

Kompresní funkce užívá následující interní funkce:

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

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

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

Zde $\wedge$ znamená bitové AND, $\vee$ je bitové OR a $\otimes$ bitové XOR.

Každý 512-bitový blok X zprávy (16 32-bitových bloků plaintextu) je zpracován následujícím algoritmem:

$X_j \leftarrow p_j$ pro $j \in 0, \dots, 15$... rozloží blok na 32bitové řetězce

for $j=16; j \leq 79; j++$ { ... rozexpaduje na 80 řetězců

$$X_j \leftarrow (X_{j-3} \otimes X_{j-8} \otimes X_{j-14} \otimes X_{j-16}) \lll 1;$$

}

$$(A, B, C, D, E) \leftarrow (H_1, H_2, H_3, H_4, H_5);$$

for ($j=0; j \leq 19; j++$) {

$$t \leftarrow (A \lll 5) + f(B, C, D) + E + X_j + y_j);$$

$$(A, B, C, D, E) \leftarrow (t, A, B \lll 30, C, D);$$

}

for ($j=20; j \leq 39; j++$) {

$$t \leftarrow (A \lll 5) + h(B, C, D) + E + X_j + y_j);$$

$$(A, B, C, D, E) \leftarrow (t, A, B \lll 30, C, D);$$

}

for ($j=40; j \leq 59; j++$)\{

$$t \leftarrow (A \lll 5) + g(B, C, D) + E + X_j + y_j);$$

$$(A, B, C, D, E) \leftarrow (t, A, B \lll 30, C, D);$$

}

for ($j=60; j \leq 79; j++$)\{

$$t \leftarrow (A \lll 5) + f(B, C, D) + E + X_j + y_j);$$

```

(A, B, C, D, E) ← (t, A, B <<< 30, C, D);
}
(H1, H2, H3, H4, H5) ← (H1 + A, H2 + B, H3 + C, H4 + D, H5 + E);

```

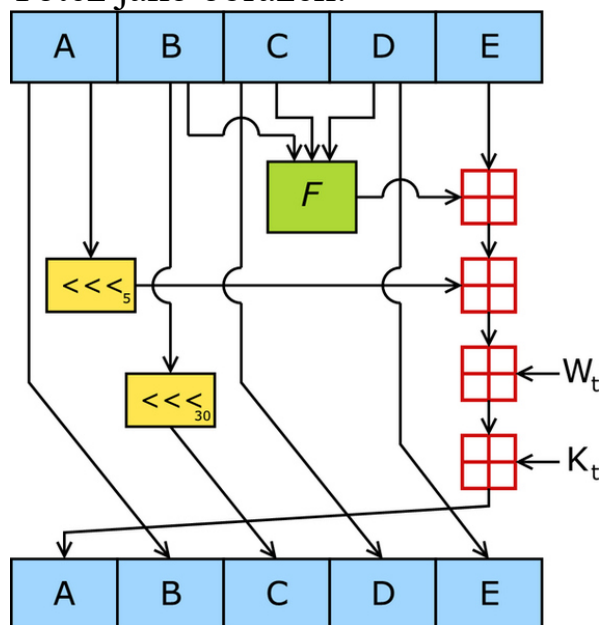
Zde $x \lll y$ znamená rotaci x o y bitů vlevo, y_j je konstanta definovaná pro dané kolo.

Dokončení

Po zpracování posledního bloku SHA-1 vyprodukuje *digest*:

digest = $H_1 \mid H_2 \mid H_3 \mid H_4 \mid H_5$

Totéž jako obrázek:



(zde y_j odpovídá K_t , tj. elementům konstantního pole 80 položek o délce 32 bitů, W_t je vstupní blok plaintextu)

Na konci roku 2005 byl oznámen útok spočívající v nalezení **kolize** se složitostí 2^{63} operací (Wang, Yao, Yao), v současné době nejrychlejší útok pro nalezení kolize je od Marca Stevensa se složitostí menší než 2^{58} . Prakticky použitelnější útok se zachováním zvoleného prefixu zprávy Leurent-Peyrin (tj. kolize je zkonstruována na omezené části zprávy) má složitost 2^{64} .

Z praktického hlediska **toto neznamena bezprostřední nutnost přestat SHA-1 používat, ale v nových implementacích by měl být upřednostněn některý z novějších algoritmů již by algoritmus neměl být používán**

SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512)

principiálně stejné algoritmy, jako SHA-1
jednotliví členové rodiny SHA-x se od sebe liší délkou slova (32, nebo 64 bitů) a počtem kol 64 u SHA-256, 80 u SHA-512
jde o nově navržené algoritmy, které by měly pokrývat dosud známé slabiny předchozí generace

Příprava konstant

```
h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19
```

$k(0..63) \leftarrow$

```
0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
```

Příprava vstupních dat

na konec dat připojit bit "1"

doplnit bity "0" do celkové délky $\equiv 448 \equiv -64 \pmod{512}$

doplnit původní délku dat v bitech jako 64-bit big-endian integer

Zpracování zprávy po 512 bitových blocích

každý blok rozložit do šestnácti 32-bitových big-endian wordů $w(i)$, $0 \leq i \leq 15$

expandovat šestnáct 32-bit wordů do šedesátičtyř 32-bit wordů:

for i from 16 to 63 do

$s_0 \leftarrow (w(i-15) \ggg 7) \oplus (w(i-15) \ggg 18) \oplus (w(i-15) \gg 3)$

$s_1 \leftarrow (w(i-2) \ggg 17) \oplus (w(i-2) \ggg 19) \oplus (w(i-2) \gg 10)$

$w(i) \leftarrow w(i-16) + s_0 + w(i-7) + s_1$

enddo

Inicializace

$$a \leftarrow h_0; b \leftarrow h_1; c \leftarrow h_2; d \leftarrow h_3; e \leftarrow h_4; f \leftarrow h_5; g \leftarrow h_6; h \leftarrow h_7$$
Hlavní smyčka

for i from 0 to 63 do

$$s_0 \leftarrow (a \ggg 2) \oplus (a \ggg 13) \oplus (a \ggg 22)$$

$$maj \leftarrow (a \wedge b) \oplus (b \wedge c) \oplus (c \wedge a)$$

$$t_0 \leftarrow s_0 + maj$$

$$s_1 \leftarrow (e \ggg 6) \oplus (e \ggg 11) \oplus (e \ggg 25)$$

$$ch \leftarrow (e \wedge f) \oplus ((\neg e) \wedge g)$$

$$t_1 \leftarrow h + s_1 + ch + k(i) + w(i)$$

enddo

$$h \leftarrow g; g \leftarrow f; f \leftarrow e; e \leftarrow d + t_1; d \leftarrow c; c \leftarrow b; b \leftarrow a; a \leftarrow t_0 + t_1$$
Přidání výsledku bloku do stavu výpočtu

$$h_0 \leftarrow h_0 + a$$

$$h_1 \leftarrow h_1 + b$$

$$h_2 \leftarrow h_2 + c$$

$$h_3 \leftarrow h_3 + d$$

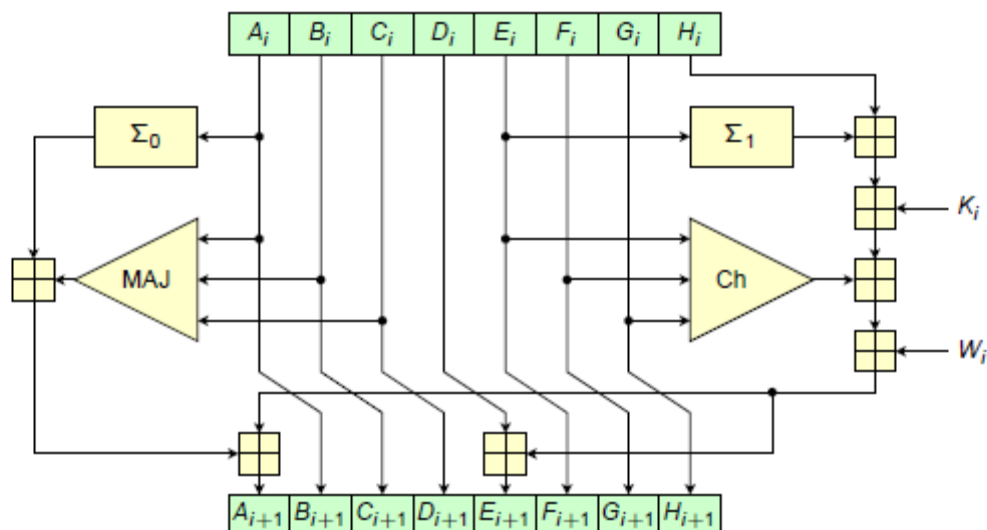
$$h_4 \leftarrow h_4 + e$$

$$h_5 \leftarrow h_5 + f$$

$$h_6 \leftarrow h_6 + g$$

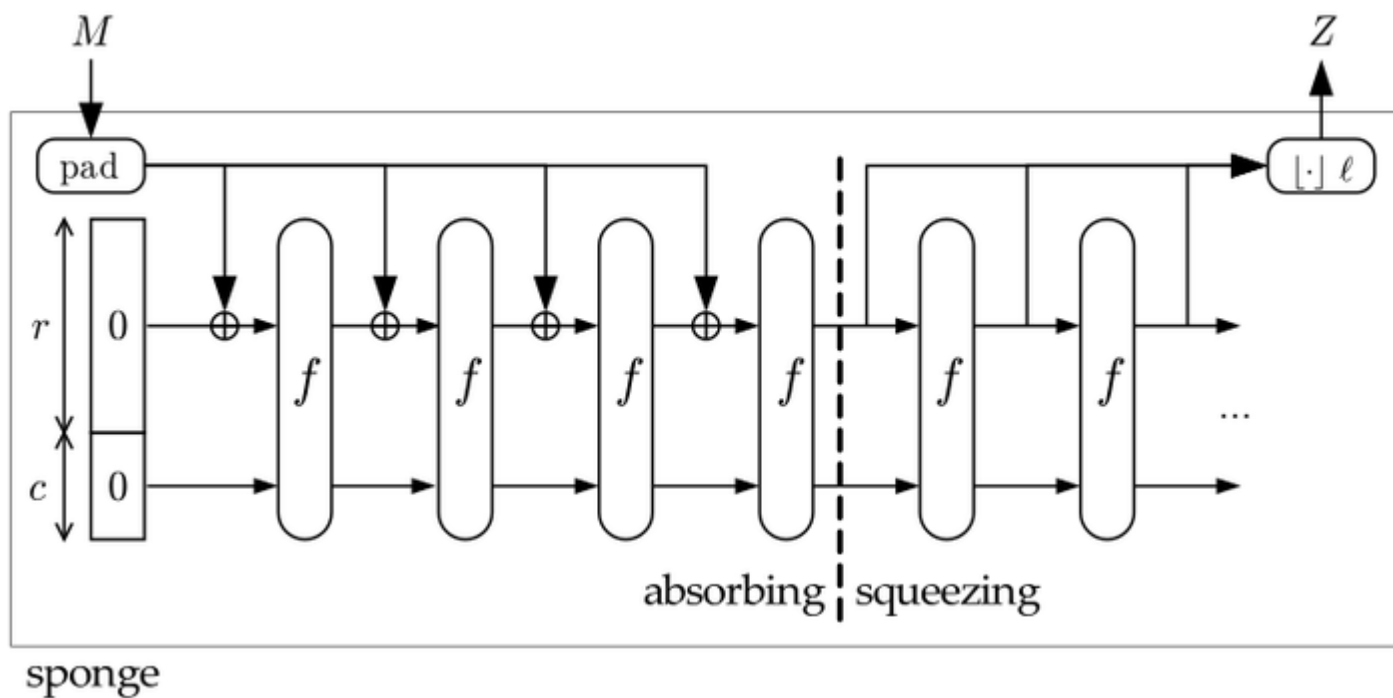
$$h_7 \leftarrow h_7 + h$$
Konstrukce výsledku:

$$\text{digest} \leftarrow h_0 \mid h_1 \mid h_2 \mid h_3 \mid h_4 \mid h_5 \mid h_6 \mid h_7$$



SHA-3

Představuje nejaktuálnější standard kryptografického hašování, jedná se o vybrané instance transformace Keccak



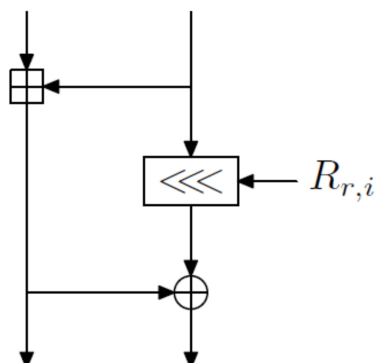
Skein

interní stav o velikosti 256, 512 nebo 1024 bitů
 založena na šifrovacím algoritmu ThreeFish a UBI módu
 v závislosti na velikosti stavu se určí počet kol:

stav	#slov	#kol
256	4	72

512 8 72
1024 16 80

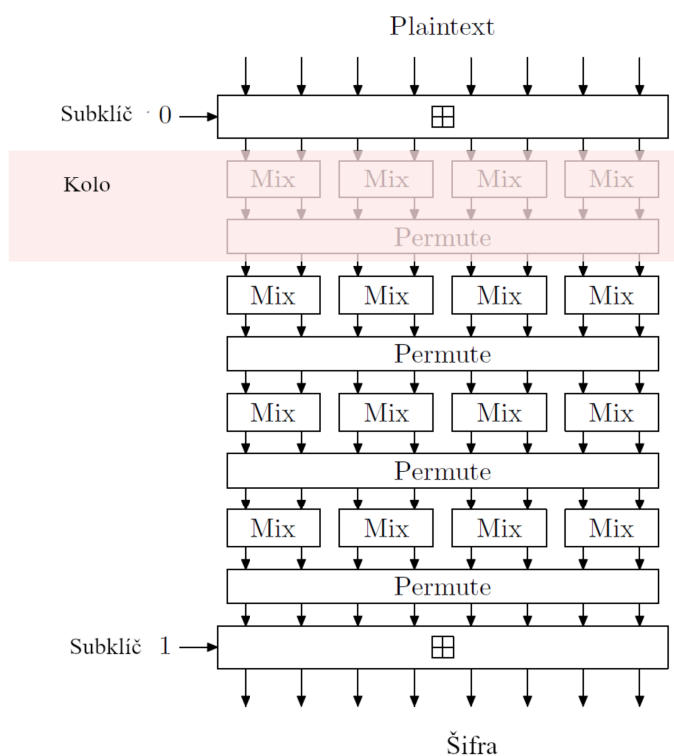
Základem kola primitivní operace Mix:



Zde:

sčítání mod 64,
rotace pevně daná tabulkou
vstupem dvě 64 bit slova

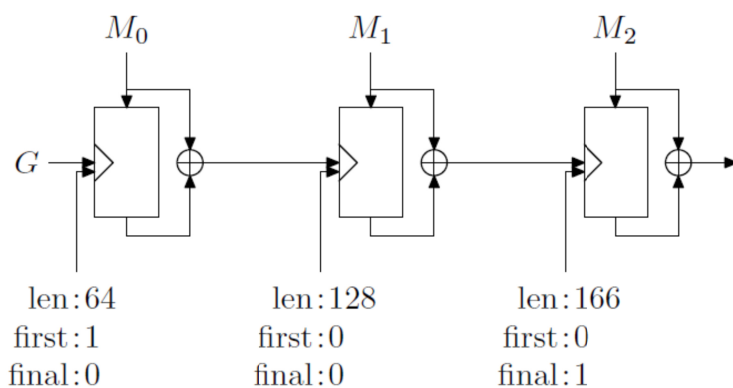
Kompresní funkce je potom realizována šifrováním TriFish algoritmem:



Zde varianta TriFish 256

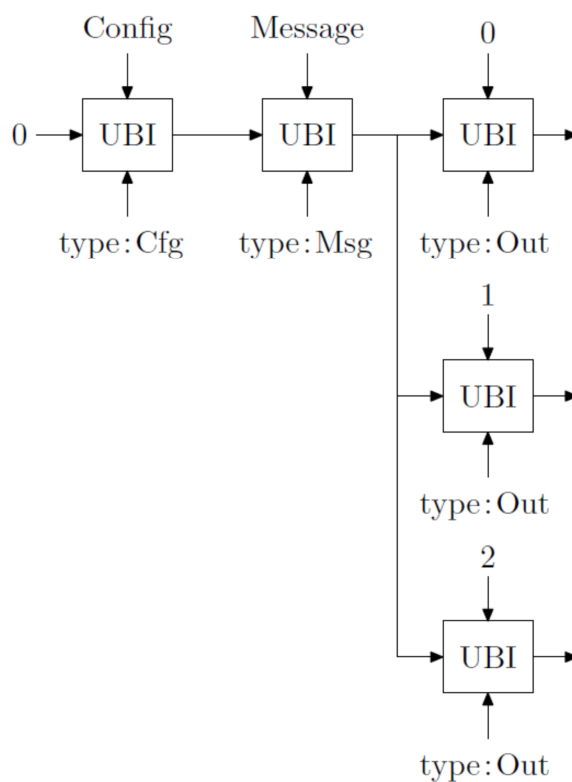
Expanze subklíčů

UBI mód šifrování

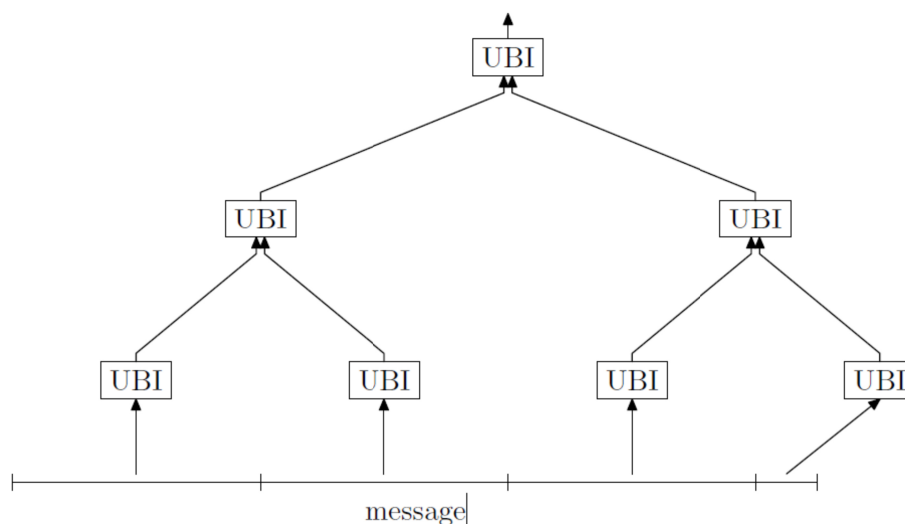


Skein hašování

opakováním UBI nejprve nad konfiguračním řetězcem, následně nad daty a poté rotace nad nulovým řetězcem pro vygenerování výstupu požadované délky



Hierarchické hashování (tree hash)



Instance algoritmu	Velikost výstupu	Rate r tj. velikost bloku	Kapacita c	Definice
SHA3-224(M)	224	1152	448	Keccak[448]($M \parallel 01, 224$)
SHA3-256(M)	256	1088	512	Keccak[512]($M \parallel 01, 256$)
SHA3-384(M)	384	832	768	Keccak[768]($M \parallel 01, 384$)
SHA3-512(M)	512	576	1024	Keccak[1024]($M \parallel 01, 512$)
SHAKE128(M, d)	d	1344	256	Keccak[256]($M \parallel 1111, d$)
SHAKE256(M, d)	d	1088	512	Keccak[512]($M \parallel 1111, d$)

Bezpečnost

preimage resistance $c/2$

nalezení kolize stejná složitost

Instance algoritmu	Bezpečnost (v bitech)		
	Kolize	Preimage	2nd preimage
SHA3-224(M)	112	224	224
SHA3-256(M)	128	256	256
SHA3-384(M)	192	384	384
SHA3-512(M)	256	512	512
SHAKE128(M, d)	$\min(d/2, 128) \geq \min(d, 128)$		

$$\text{SHAKE256}(M, d) \quad \min(d/2, 256) \geq \min(d, 256) \min(d, 256)$$

Návrhy MAC funkcí

obecná struktura stejná jako MDC, funkce f a případně i IV závisí navíc na klíči K
existuje velmi malé množství algoritmů

nejčastěji používanou metodou je počítání vhodné symetrické šifry v módu CBC, resp CFB, konkrétní schémata se liší volbou případně kombinací zmíněných módů, použitím různých zarovnání ...

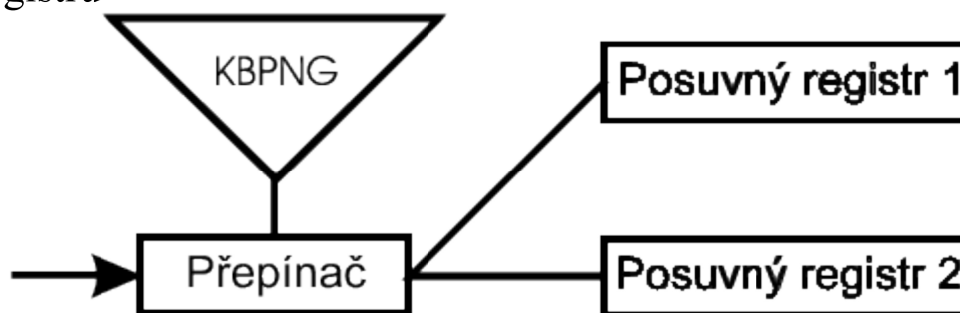
Jinou možností je výpočet

$$f = E(K, X_i \oplus H_{i-1}) \oplus X_i$$

Uvedené stále platí, ale dnes bychom to dělali jinak (viz dále)

Stream MAC

potřebujeme kryptograficky bezpečný generátor pseudonáhodných sekvencí
podle výsledku generátoru je vstupní bit přesunut do prvního nebo druhého posuvného registru se zpětnou vazbou, výsledek určen konečným obsahem posuvných registrů



klíčem očividně seed generátoru

HMAC

vlastně obecný návod jako zkonstruovat MAC na základě jakékoliv hashovací (MDC) funkce:

$$\text{HMAC}_K(m) = h((K \oplus \text{opad}) \| h((K \oplus \text{ipad}) \| m)),$$

kde oba bloky paddingu jsou definovány jako konstanty $\text{opad} = 0x5c5c5c...5c$,
 $\text{ipad} = 0x363636...36$

Rozbor nabízených a požadovaných vlastností hašovacích funkcí pro různé aplikace:

Vlastnosti	preimage	2nd preimage	odolnost proti kolizi
MDC+ asym. podpis	ano	ano	ano
MDC+ autentický kanál		ano	ano
Uložení hesel (MDC)	ano		
MAC (neznámý klíč)	ano	ano	ano

Generátory pseudonáhodných sekvencí

všechny tyto generátory jsou **deterministickými** p-time algoritmy, které na vstupu přijmou náhodný řetězec a expandují jej do (obvykle) mnohem delší posloupnosti

Definice:

Náhodná posloupnost je taková posloupnost, kterou nelze generovat programem kratším, než je ona sama.

Definice:

Pseudonáhodnou posloupností budeme rozumět posloupnost délky n , o které žádný deterministický p-time algoritmus není schopen s pravděpodobností větší než $(1/n) + \epsilon$ rozhodnout, zda se jedná o náhodnou posloupnost, či nikoliv.

Kongruenční generátor

Lineární

$$X_i = (aX_{i-1} + b) \bmod m$$

při vhodné volbě a , b , m (např. m prvočíslo) generuje neopakující se posloupnost délky m - generátor s maximální délkou

kvadratický

$$X_i = (aX_{i-1}^2 + bX_{i-1} + c) \bmod m$$

kubický

$$X_i = (aX_{i-1}^3 + bX_{i-1}^2 + cX_{i-1} + d) \bmod m$$

kongruenční generátory jsou velmi rychlé, avšak predikovatelné, byla vypracována analýza

Posuvné registry s lineární zpětnou vazbou

(linear feedback shift registers)

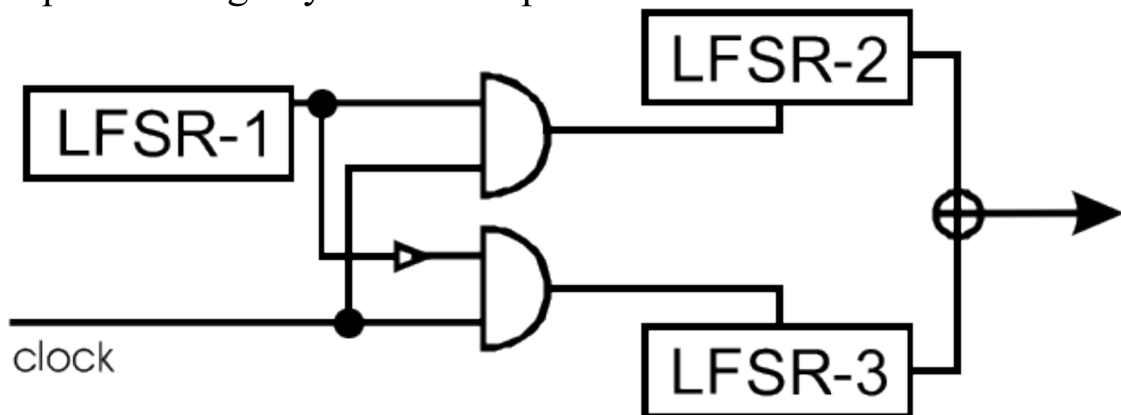
sestávají z posuvného registru a vypouštěcí sekvence (tap sequence), což je polynom stupně max. délky posuvného registru

v každém kroku je obsah registru posunut o bit doprava, výstupem je nejpravější bit, registr se zleva doplní o XOR těch bitů v registru, které odpovídají koeficientům vypouštěcího polynomu

pro generátor s maximální délkou nutno polynom volit primitivní polynom stupně n - ireducibilní polynom dělící $x^{2^n-1} + 1$, který nedělí $x^d + 1$ pro lib d dělící $2^n - 1$

Střídající stop-and-go generátor

používá tři posuvné registry s lineární zpětnou vazbou



impuls hodin jsou v závislosti na hodnotě LFSR-1 přiveden na LFSR-2 nebo LFSR-3, výstup těchto registrů je XORován, čímž vzniká výsledná hodnota. generátor má velmi dlouhou periodu

Blum-Micali

$$X_i = a^{X_{i-1}} \bmod p$$

a, p prvočísla, bezpečnost plyne z obtížnosti výpočtu diskretního logaritmu

RSA generátor

$$X_i = X_{i-1}^e \bmod n$$

výsledkem nejnižší bit X_i . Bezpečnost ekvivalentní s bezpečností RSA

Blum Blum Shub (BBS)

$$X_i = (X_{i-1}^2) \bmod n$$

kde n je Blumovo číslo - součin dvou velkých prvočísel kongruentních s 3 mod 4. Generátor se inicializuje hodnotou $X_0 = (X^2) \bmod n$, kde X je náhodné číslo nesoudělné s n .

Výsledkem je nejnižší bit X_i . Bezpečnost vyplývá z obtížnosti faktorizace velkých čísel. Generátor je nepredikovatelný vlevo i vpravo, ale je možné se znalostí faktorů p, q čísla n spočítat přímo libv. prvek generované posloupnosti!

$$X_i = X_0^{(2^i) \bmod ((p-1)(q-1))} \bmod n$$

Dosud není známa metoda zlomení tohoto generátoru. Nevýhodou značná pomalost.

Generátor Yarrow

uvádím jako příklad komplexní konstrukce náhodného generátoru
cílem útočníka je získat vnitřní stav generátoru, pokud uspěje, může predikovat výstup generátoru, nebo rekonstruovat dřívější části výstupní posloupnosti
kvalita generátoru závisí na kvalitě vstupu, častým problémem nesprávný odhad míry entropie zdrojů

generátor se skládá ze čtyř hlavních komponent:

- akumulátor entropie (entropy accumulator)
- mechanismus změny klíče (reseed mechanism)
- mechanismus generování (generation mechanism)
- řízení změny klíče (reseed control)

cílem je kvalitní akumulace entropie ze všech zdrojů, zajištění dostatečně častých (nealgoritmických !) změn klíče a občasná změna klíče, která bude pro útočníka nepřekonatelná i v případě, že u některého zdroje došlo k nadhodnocení obsahu entropie



Akumulátor entropie

používají se dva pooly – rychlý a pomalý

každý pool je realizován jako kontext hashovací funkce (zde SHA-1), do kterého se přimíchávají vstupy „tak jak přicházejí“

Mechanismus změny klíče

na pokyn řízení změny klíče počítá novou hodnotu klíče K

změna klíče na základě rychlého poolu využívá

- stávající klíč
- hash všech stavů v rychlém poolu od poslední změny klíče

změna klíče na základě pomalého poolu využívá

- stávající klíč
- hash všech vstupů v rychlém poolu od poslední změny klíče
- hash všech vstupů v pomalém poolu od poslední změny klíče

po změně klíče jsou vynulovány čítače entropie všech dotčených poolů

Řízení změny klíče

s každým zdrojem entropie je v poolu udržován čítač nashromážděných náhodných bitů

- změna klíče z rychlého poolu se provede pokud aspoň jeden čítač přesáhne práh
 - změna klíče z pomalého poolu se provede pokud aspoň k čítačů přesáhne práh
- práh pro rychlý pool bývá 100, pro pomalý 160, k se v závislosti na prostředí volí $2 \div 3$

Mechanismus generování

využívá se symetrická šifra (3DES)

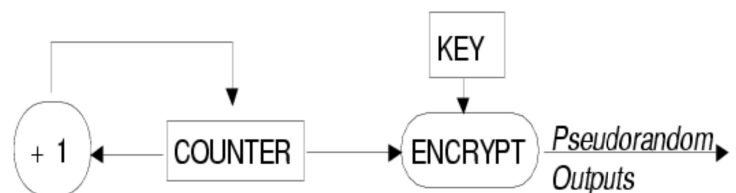
běžící v tzv. counter módu

vstupem pro šifru je obsah čítače C a klíč K :

$$C \leftarrow (C + 1) \bmod 2^n$$

$$\text{výstup} \leftarrow E_K\{C\}$$

mechanismus generování nejpozději po $1 \leq P_g \leq 2^{n/3}$ bitech interně vygeneruje $|K|$ bitů a použije je jako nový klíč – jde o tzv. generator gate, která zabraňuje útočníkovi počítat „do minulosti“



uvedená konstrukce je možná i s použitím jiných kryptografických primitiv

Generátory náhodných posloupností

... produkují skutečně náhodné posloupnosti

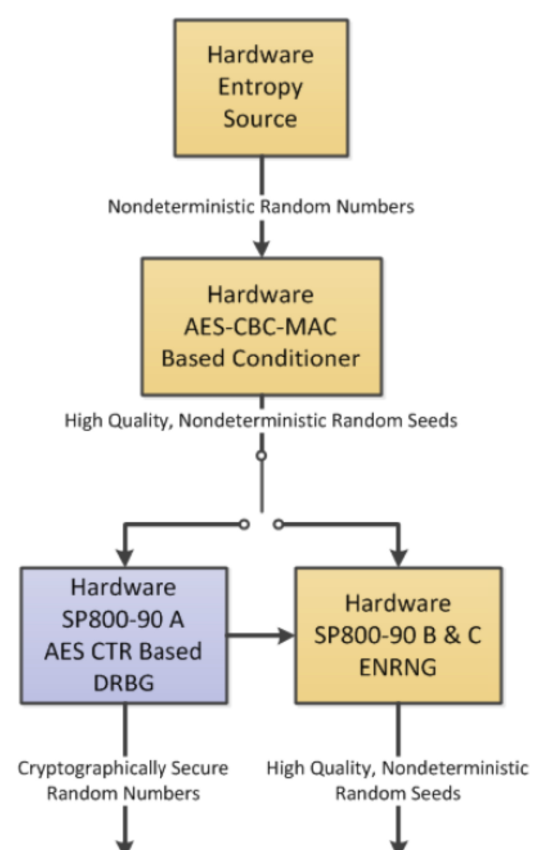
Intel DRNG

založen na HW zdroji entropie – snímání termálního šumu polovodiče s frekvencí cca 3GHz

následuje kondicioner – AES běžící v CBC-MAC módu zajišťuje akumulaci a vyrovnaní toku dat

finální bloky

– deterministický generátor inicializovaný (seeded) s kondicionérem



- nedeterministický generátor prezentující přímý výstup z kondicionéru

součástí řešení odpovídající knihovny pro extrakci dat z finálních stupňů

na obrázky chybí stupeň kontroly healthtest, který provádí statistickou kontrolu výstupu ES před vstupem do kondicionéru

Generátory založené na speciálním hardware

odvíjejí svoji činnost od sledování nějakého fyzikálního jevu, který je ve své podstatě nepredikovatelný, zhusta dosti sofistikovaně „nevhodným“ způsobem:

- měření záření kousku radioaktivního materiálu
- termální šum polovodiče
- měření stavu volně běžícího oscilátoru
- vnitřní šum zesilovače
- samovolné vybíjení kondenzátoru
- ...
- dnes již běžně dostupné generátory jako součást procesoru

Generátory softwarové

jsou založeny na pozorování jevů v počítači, které jsou z hlediska programu náhodné:

- doba odezvy diskového systému
- čas mezi stisky kláves uživatelem
- pohyb myši
- systémový časovač
- stav vnitřních tabulek OS
- statistiky síťové komunikace
- ...

zásadním problémem odhad množství a rozložení entropie v generovaných datech dnes v době virtualizace nutno opatrně posuzovat jejich spolehlivost

Testy náhodných generátorů

Monobit test

zda počet „1“ a „0“ je přibližně ‚správný‘

$$X_1 = \frac{(n_0 - n_1)^2}{n}$$

zde n_0 je počet „0“, n_1 je počet „1“ ve vzorku délky n .

Serial test

zda počet výskytů „00“, „01“, „10“ a „11“ je přibližně stejný, jednotlivé podřetězce se mohou překrývat.

$$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1$$

zde n_{xx} výskytů řetězce „xx“; $n_{00} + n_{01} + n_{10} + n_{11} = n - 1$

Poker test

zkoumá, zda posloupnost obsahuje zhruba stejný počet všech možných podposloupností délky m

buď m tž. $\left\lfloor \frac{n}{m} \right\rfloor \geq 5(2^m)$ a položíme $k = \left\lfloor \frac{n}{m} \right\rfloor$

vstupní posloupnost rozdělíme na k nepřekrývajících se podposloupností délky m , označíme n_i počet výskytů i -tého typu podposloupnosti

$$X_3 = \frac{2^m}{k} \left(\sum_{i=1}^{2^m} n_i^2 \right) - k$$

Runs test

zda vstupní posloupnost obsahuje správný počet běhů, tj. posloupností samých nul (díry) a jedniček (bloky); očekávaný počet běhů délky i je $e_i = (n - i + 3)/2^{i+2}$

nechť k je nejvyšší hodnota i tž. $e_i \geq 5$.

$$X_4 = \sum_{i=1}^k \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^k \frac{(G_i - e_i)^2}{e_i}$$

Zde B_i je počet bloků, G_i počet děr délky i .

Autocorellation test

cílem je srovnání korelace mezi vstupní posloupností s a její posunutou verzí

buď d tž. $1 \leq d \leq \lfloor n/2 \rfloor$, počet bitů, ve kterých se s liší od posunuté verze označíme

$$A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$$

$$X_5 = 2 \left(A(d) - \frac{n-d}{2} \right) / \sqrt{n-d}$$

na základě uvedených kritérií je následně možno definovat sady statistických testů, např. **FIPS140-1 test náhodnosti**:

vstupem testu je výstupní posloupnost generátoru o délce 20 000 bitů, která musí splnit všechna čtyři kritéria

1. monobit test: s požadavkem $9654 < X_1 < 10346$

2. poker test: X_3 počítané pro $m = 4$ musí vyhovovat $1,03 < X_4 < 57,4$

3. runs test: počítají se běhy délky 1 až 6 (všechny delší běhy se počítají jako by měly délku 6), pro všech 12 druhů běhů musí platit

délka	min	max
1	2267	2733
2	1079	1421
3	502	748
4	223	402
5	90	223
6	90	223

4. long run test: žádný běh delší než 34

tato sada testů musí být aplikována při každém spuštění generátoru

V následující verzi FIPS140-2 došlo k zúžení mezí a následně byl celý test označen za optional