III.

Mody činnosti blokových šifer a hašovací funkce

Vlastimil Klíma

verze: 2.1, 11.4.2007

Obsah

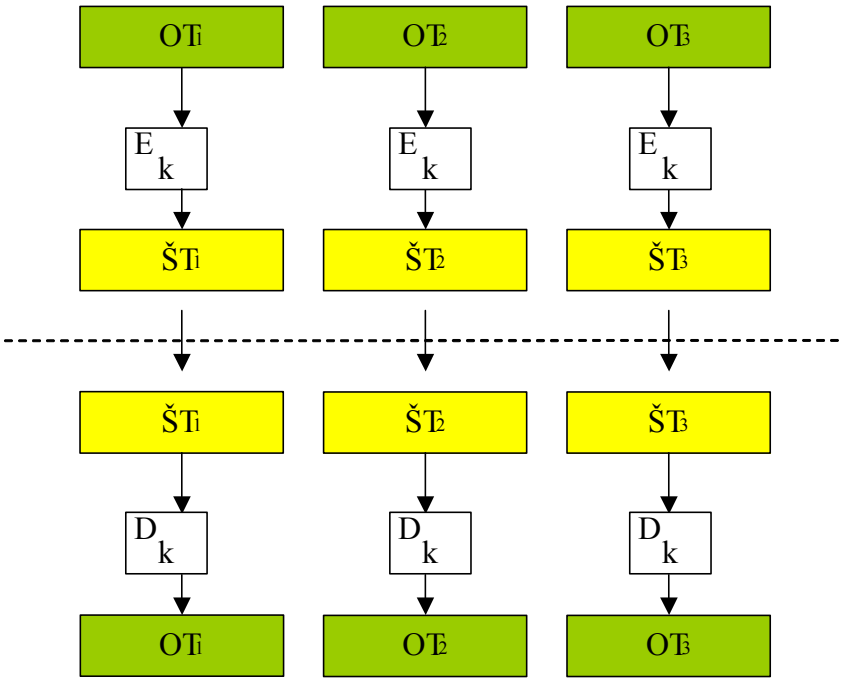
11. Operační mody blokových šifer

V praxi se setkáváme s požadavky šifrovat nejen celé *N*-bitové bloky, ale obecně jakoukoliv posloupnost bitů. Pokud máme k dispozici pouze *N*-bitovou blokovou šifru, musíme ji umět použít k šifrování jakékoliv délky posloupnosti bitů, extrémně na jeden bit. Proto vznikly operační mody blokových šifer. Operační mody blokových šifer jsou způsoby použití blokových šifer v daném kryptosystému.

Pomocí operačních modů můžeme získat nové zajímavé vlastnosti a využití blokových šifer. Seznámíme se s mody ECB, CFB, OFB, CBC, CTR a MAC.

11.1. Elektronická kódová kniha (ECB)

Bloková šifra je jednoduchou substitucí, přičemž její bezpečnost je dána tím, že její tabulka je extrémně velká. Například u moderních šifer s délkou bloku $N = 128$ bitů má tato tabulka 2^{128} položek. V tomto případě nejsme dokonce schopni tabulku ani celou vypočítat ani uložit. Ukládá se pouze algoritmus výpočtu, nikoli tabulka této transformace.
Pokud otevřený text rozdělíme na bloky OT_1, OT_2,... OT_n a každý šifrujeme zvlášť jako $ŠT_i = E_k(OT_i)$, i = 1, 2, ..., n, tento operační modus se nazývá elektronická kódová kniha (ECB, Electronic Codebook). Je to základní modus a prakticky se využívá k šifrování dat velmi málo.



Obr.: Modus ECB

11.1.1. Informace, vyzařující ze šifrového textu v modu ECB

I když luštění substituce nad extrémně velkou abecedou je diametrálně odlišné od luštění substituce nad abecedou o 26 znacích, zůstávají nevýhody klasické substituce zachovány iu blokové šifry s velkou substituční tabulkou. Nevýhodou takového způsobu šifrování totiž je, že **stejné bloky otevřeného textu mají vždy stejný šifrový obraz**. Pokud nalezneme několik shodných bloků šifrového textu, víme, že skrývají stejný otevřený text. V určitém kontextu to může dokonce rozkrývat i **hodnotu** otevřeného bloku nebo jiné informace. Například pokud

jsou prázdné sektory na pevném disku PC vyplněny hodnotou 0xFF (nebo jinou shodnou), lze ze šifrového obrazu disku jasně vidět, kde jsou uložena data, a kde jsou hodnoty 0xFF. Lze například zjistit, zda a jak dlouhá zpráva byla na disk nově zapsána. Podobně vyzařují informace i šifrované soubory, které mají hodně shodných bloků a informace je skryta v řídkých změnách (například obrázek bílého kruhu na černém pozadí bude přímo vidět jako silueta i v šifrovém textu). Šifrování v modu ECB se proto používá tam, kde jako otevřený text vystupují náhodné binární řetězce.

11.1.2. Integrita a modus ECB

Ve zprávě, šifrované modem ECB, **může útočník také bloky šifrového textu vyměňovat, vkládat nebo vyjímat**, a tak snadno docilovat pro uživatele nežádoucích změn v otevřeném textu, zejména, pokud nějakou dvojici (OT, ŠT) už zná. Tím je opět ilustrován v praxi často opomíjený fakt, že integrita otevřeného textu se šifrováním nezajistí.
Na obrázku vidíme, že vložením šifrového bloku se docílí změna otevřeného textu, aniž je nutné znát šifrovací klíč.

```
.......  3tdszj34  j7čžuths  bgžc4rš7  rg43č7řz  ......
..............  převeďte  1  0 0 0  ,- Kč  ......

.......  3tdszj34  j7čžuths  bgžc4rš7  bgžc4rš7  rg43č7řz  ......
..............  převeďte  1  0 0 0  0 0 0  ,- Kč  ......
```

Obr.: Útok na modus ECB vložením bloku šifrového textu

11.2. Řetězení šifrového textu (CBC)

11.2.1. Rozšíření difúze a konfúze z bloku na celý otevřený text

Moderní blokové šifry dosahují velmi dobrých vlastností jak difúze, tak konfúze, ale pouze v rámci jednoho bloku otevřeného textu. Protože otevřeným textem je v praxi většinou mnoho bloků, zavádí se modus řetězení šifrového textu CBC, Cipher Block Chaining). Každý blok otevřeného textu se v něm nejprve modifikuje předchozím blokem šifrového textu, a teprve poté se šifruje. Protože šifrový text by měl být náhodný, stává se následně modifikovaný otevřený text také náhodným. To odstraňuje nevýhody modu ECB.

Tímto způsobem běžný šifrový blok závisí na celém předchozím otevřeném textu z důvodu řetězení této závislosti přes předchozí šifrový text. Nevýhodou (z hlediska difúze a útoků) a současně výhodou (z hlediska samosynchronizace) je, že tato závislost se do běžného šifrového bloku zavádí prostřednictvím pouze předchozího šifrového bloku.

11.2.2. Definice CBC

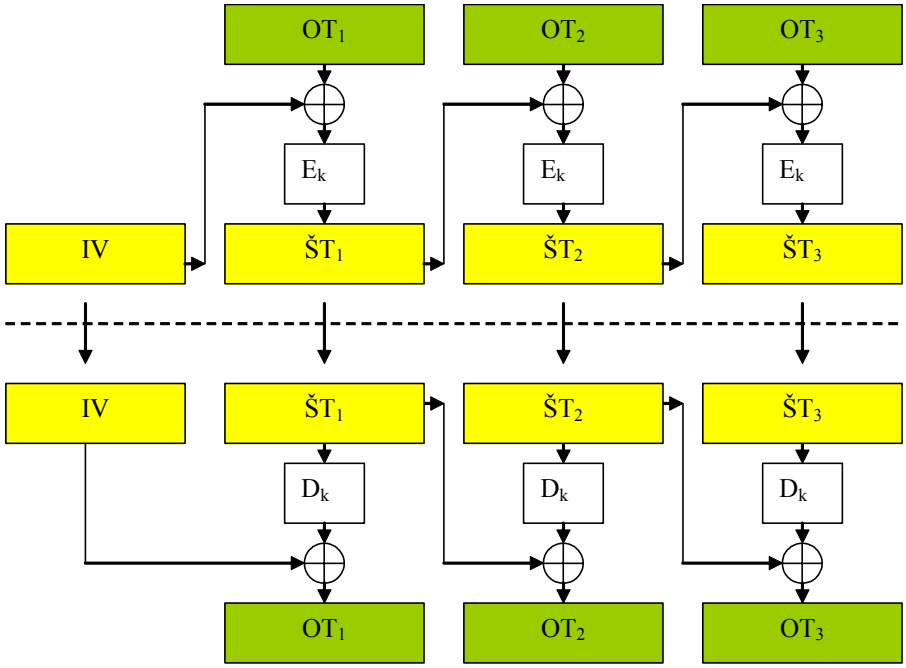
CBC je v současné době nejpoužívanějším operačním modem blokových šifer. Eliminuje některé slabosti modu ECB a zajišťuje difúzi celého předchozího otevřeného textu do daného bloku šifrového textu. První blok otevřeného textu je modifikován náhodnou, tzv. inicializační hodnotou (initializing value, IV), která je vysílána před vlastním šifrovým textem, podobně jako u proudových šifer. Šifrování se provádí podle vztahů

$ŠT_0 = IV,$
$ŠT_i = E_k(OT_i \oplus ŠT_{i-1}), i = 1, 2, ..., n.$

a odšifrování probíhá podle vztahů:
$ŠT_0 = IV,$

$OT_i = ŠT_{i-1} \oplus D_k(ŠT_i)$, i = 1, 2, ..., n.

Pokud bychom dvakrát šifrovali shodný první blok OT_1, různé náhodné inicializační vektory ho modifikují na různé náhodné vstupní bloky. Výsledný blok šifrového textu $ŠT_1$ je proto také náhodný a různý. Protože však tento blok přebírá úlohu inicializačního vektoru pro šifrování následujícího bloku otevřeného textu, efekt znáhodnění se postupně projeví i na dalších blocích. Odtud vyplývá, že **budeme-li šifrovat jeden a tentýž, byť i velmi dlouhý, otevřený soubor dat dvakrát v modu CBC, obdržíme odlišný šifrový text.**



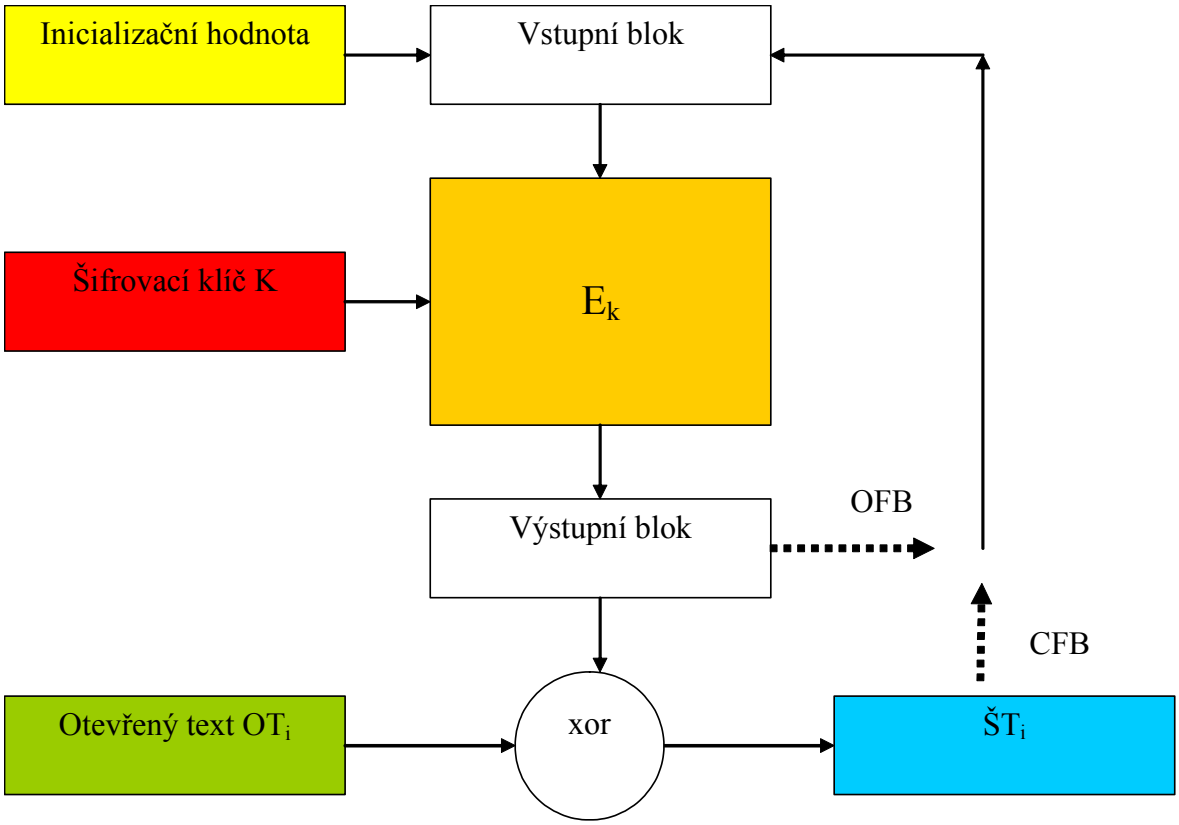
Obr.: Modus CBC

11.2.3. Samosynchronizace modu CBC

Z definice modu CBC vyplývá vlastnost samosynchronizace, kterou jsme viděli i u proudových šifer. Proces odšifrování je schopen se zotavit a produkovat správný otevřený text i při výpadku nebo porušení několika bloků šifrového textu. K obnově otevřeného textu dojde už při dvou za sebou jdoucích správných blocích ŠT. Z obrázku i z rovnice pro odšifrování vidíme, že ke správnému odšifrování bloku OT_i potřebujeme pouze správné bloky šifrového textu $ŠT_{i-1}$ a $ŠT_i$.

11.3. Mody zpětné vazby ze šifrového textu (CFB) a z výstupu (OFB)

Tyto operační mody převádí blokovou šifru na proudovou, přesněji používají blokovou šifru k vygenerování hesla, které se binárně načítá (xoruje) na otevřený text. Používají náhodnou inicializační hodnotu IV k nastavení odpovídajícího konečného automatu do náhodné polohy. Tento automat pak produkuje posloupnost hesla, které se jako u proudových šifer xoruje na otevřený text. První blok hesla se získá zašifrováním IV. Konečný automat pracuje tak, že vzniklé heslo (v modu OFB, Output Feedback) nebo vzniklý šifrový text (v modu CFB, Cipher Feedback) vede na vstup blokové šifry a jeho zašifrováním je produkován následující blok hesla. OFB má vlastnost čisté (synchronní) proudové šifry, neboť heslo je generováno zcela autonomně bez vlivu otevřeného a šifrového textu. CFB je kombinací vlastností CBC a proudové šifry. Jako u CBC i zde k samosynchronizaci postačí dva za sebou jdoucí správné bloky šifrového textu.



Obr.: Modus zpětné vazby z výstupu (OFB) a ze šifrového textu (CFB)

CFB

zašifrování: $ŠT_0 = IV, ŠT_i = OT_i \oplus E_k(ŠT_{i-1}), i = 1, 2, ..., n.$

odšifrování: $ŠT_0 = IV, OT_i = ŠT_i \oplus E_k(ŠT_{i-1}), i = 1, 2, ..., n.$

OFB

zašifrování: $H = IV = ŠT_0,$ pro $i = 1, 2, ..., n$: $\{H = E_k(H), ŠT_i = OT_i \oplus H\}$

odšifrování: $H = IV = ŠT_0,$ pro $i = 1, 2, ..., n$: $\{H = E_k(H), OT_i = ŠT_i \oplus H\}$

Povšimněte si, že **v modech OFB a CFB se bloková šifra používá jen jednosměrně** tj. jen transformace E_k. Transformace D_k není vůbec použita. To je výhodné při hardwarové realizaci.

11.3.1. Samosynchronizace, neúplná zpětná vazba a délka periody

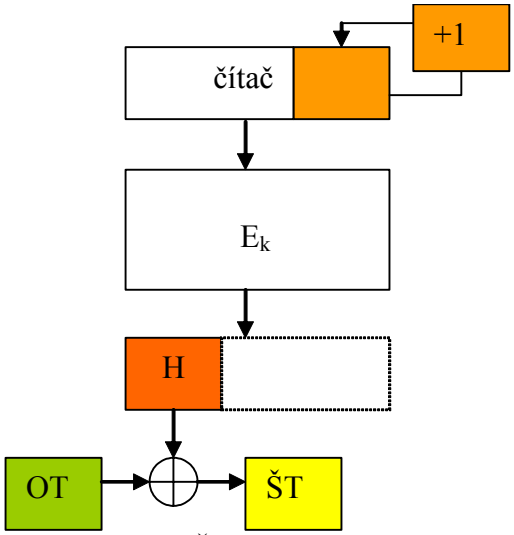
Poznamenejme, že z výstupu nemusíme použít celý blok hesla, ale jen jeho část, například *b* bitů. V tom případě se *b* bitů hesla (u OFB) nebo *b* bitů vzniklého šifrového textu (u CFB) vede zprava do vstupního registru, přičemž původní obsah vstupního registru se posune doleva o *b* bitů (*b* bitů nejvíce vlevo z registru vypadne).

Modus CFB má vlastnost samosynchronizace, a to podle délky zpětné vazby. Je-li *b* bitů, pak postačí $n + b$ nenarušených bitů šifrového textu za sebou, aby se otevřený text sesynchronizoval.

Modus OFB poskytuje synchronní proudovou šifru. Heslo generuje konečným automatem, který má maximálně 2^N vnitřních stavů. Po tomto počtu kroků se produkce hesla musí nutně opakovat. **Délka periody hesla** je proto maximálně 2^N bloků, její konkrétní délka je určena hodnotou IV a může se pohybovat náhodně v rozmezí od jedné do 2^N. Struktura hesla je značně závislá na tom, zda zpětná vazba je plná (b = N) nebo menší. Pro jakékoliv b < N je střední hodnota délky periody pouze cca $2^{N/2}$, zatímco pro b = N je to 2^{N-1}.

11.4. Čítačový modus (CTR)

Čítačový modus (CTR, Counter Mode) je v principu velmi podobný modu OFB a také převádí blokovou šifru na synchronní proudovou šifru. Odstraňuje problém s neznámou délkou periody hesla, neboť zde je délka periody hesla dána předem, a to periodou čítače. Bloková šifra jakožto bijektivní zobrazení zobrazí rozdílné hodnoty čítače na rozdílné hodnoty bloků hesla, což zajistí maximální periodu této heslové posloupnosti.



Obr.: Čítačový modus

CTR také využívá inicializační hodnotu IV, která se načte do vstupního registru (čítače) T. Po jeho zašifrování vzniká první blok hesla. Poté doje k aktualizaci čítače T, nejčastěji přičtením jedničky (odtud název modu) a ke generování dalšího bloku hesla. Heslo se může využít v plné šíři bloku nebo jen jeho b < N bitů.

Způsob aktualizace čítače je definován poměrně volně, inkrementovat se může jen například dolních B bitů čítače T. Musí se však dodržet zásada, aby ani v jedné zprávě, ani v dalších zprávách šifrovaných tímtéž klíčem, nedošlo k vygenerování stejného bloku hesla dvakrát, tj. musí se zabránit tomu, aby byl obsah čítače stejný. V takovém případě by došlo k dvojímu použití téhož hesla, což by mohlo vést k rozluštění otevřeného textu obou dotčených zpráv.

CTR
zašifrování:

$CTR_i = (IV + i - 1) \bmod 2^B$, $H_i = E_k(CTR_i)$, $ŠT_i = OT_i \oplus H_i$, i = 1,2,...

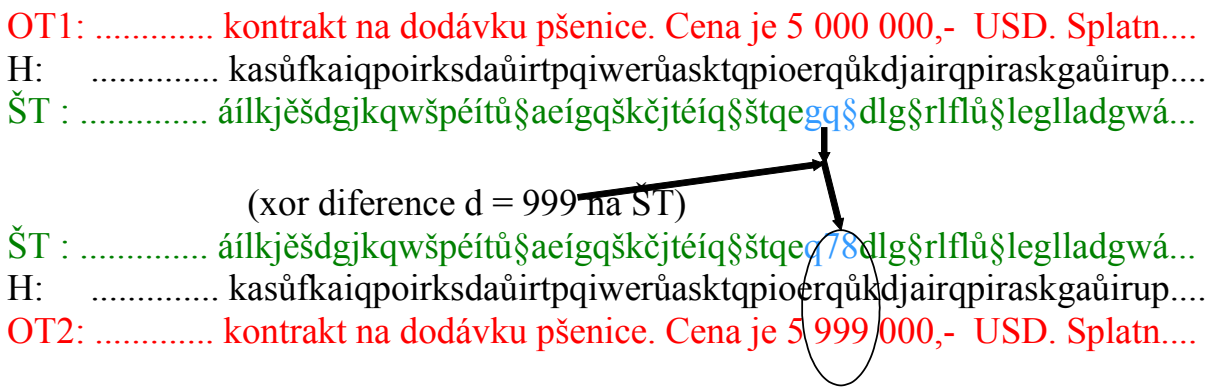
odšifrování:

$CTR_i = (IV + i - 1) \bmod 2^B$, $H_i = E_k(CTR_i)$, $OT_i = ŠT_i \oplus H_i$, i = 1,2,...

Jedná se o čistou (synchronní) proudovou šifru, používající pouze transformaci E_k i při dešifrování. Výstupní blok lze použít celý nebo jen jeho část. Smyslem modu je zaručit maximální periodu hesla, což je zaručeno periodou čítače. Další výhodou je, že **heslo může být vypočítáno jen na základě pozice otevřeného textu** a IV, nezávisle na ničem jiném. Tuto vlastnost má i modus OFB, ale než se u něj vypočítá heslo, příslušné například k miliontému bloku šifrového textu, je potřeba provést milion šifrovacích transformací E_k. Naproti tomu u modu CTR se vypočte hodnota čítače (zde counter = 1000000) a provede se jen jedna transformace E_k: E_k(counter). Možnost vypočítat heslo jen na základě pozice daného bloku v otevřeném nebo šifrovém textu je výhodná například v takových systémech, které poskytují jen danou část otevřeného nebo šifrového textu a nikoli jeho okolí. Modus CFB se liší tím, že potřebuje předchozí blok šifrového textu. CTR naproti tomu zase nemá vlastnost samosynchronizace a jakýkoliv výpadek šifrového textu znamená chybné odšifrování od tohoto místa do konce otevřeného textu.

11.5. Útoky na synchronní proudové šifry a mody CFB, OFB a CTR

Synchronní proudové šifry a blokové šifry v proudových modech CFB, OFB a CTR jsou náchylné na útok změnou šifrového textu. Změna v šifrovém textu (ŠT xor d) se promítá do otevřeného textu jako (OT xor d), tedy velmi přímočaře. Přitom diference d může být aplikována na jakémkoliv místě a v jakékoliv šíři. Toho lze využít k různým útokům, princip ilustruje obrázek.



dešifrováním změněného šifrového textu (ŠT1 ⊕ d) obdržíme
(ŠT1 ⊕ d) ⊕ H = (OT1 ⊕ H ⊕ d) ⊕ H = OT1 ⊕ d

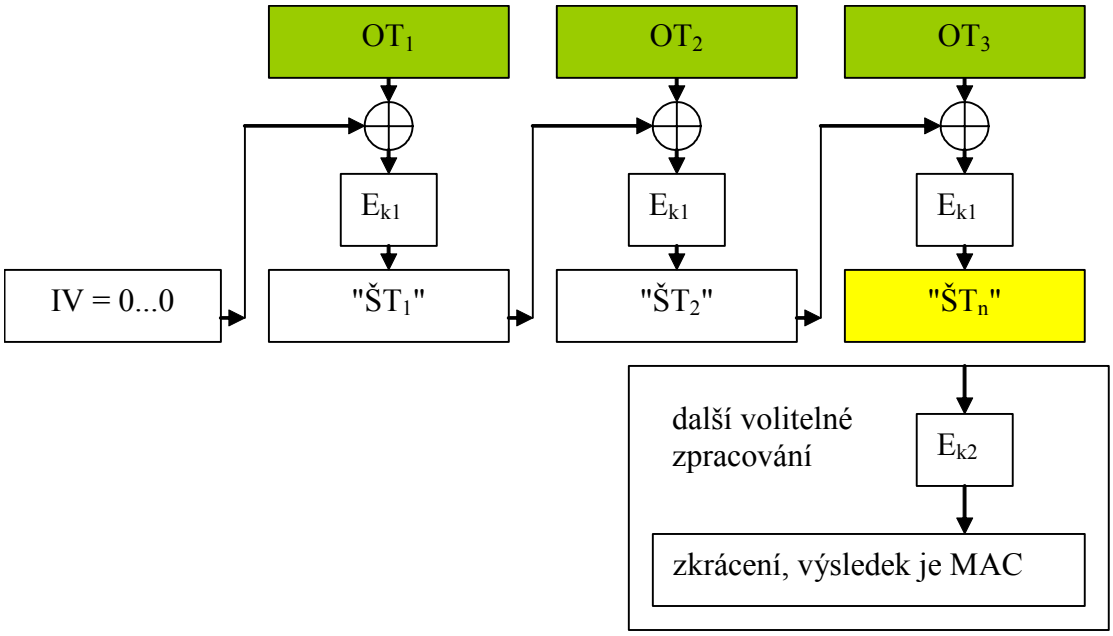
Obr.: Příklad útoku na proudovou šifru změnou šifrového textu

11.6. Autentizační kód zprávy (MAC)

Jak jsme ukázali výše, proudové a blokové šifry zajišťují důvěrnost, ale nikoli **integritu** zpráv. Mody CBC a CFB sice způsobí mírnou propagaci chyby (chyba v jednom bloku šifrového textu naruší dva bloky otevřeného textu), ale v systémech, kde není ve vlastním otevřeném textu zajištěna nějaká redundance, nemusí být tato chyba pozorována a příslušným automatizovaným systémem mohou být zpracována chybná data. **Autentizační kód zprávy** (MAC, Message Authentication Code) je dalším modem blokové šifry, který řeší právě zajištění neporušenosti dat. Tento zabezpečovací kód autentizuje původ zprávy a řeší obranu proti náhodným i úmyslným změnám nebo chybám na komunikačním kanálu. MAC je krátký

kód, který vznikne zpracováním zprávy s tajným klíčem (k1). Klíč by se měl použít jiný, než k šifrování zprávy.

Výpočet MAC probíhá tak, že se zpráva jakoby šifruje v modu CBC s nulovým inicializačním vektorem, přičemž průběžný šifrový text se nikam neodesílá. MAC je pak tvořen až posledním blokem $ŠT_n$, přičemž je možné ještě jedno přídavné šifrování navíc, tj. $MAC = E_{k2}(ŠT_n)$. Z výsledného bloku se obvykle bere jen určitá část o délce potřebné k vytvoření odolného zabezpečovacího kódu.



Obr.: Autentizační kód zprávy MAC

Pokud ke zprávě, ať otevřené nebo šifrované, připojíme MAC, příjemce může ověřit, že data pochází od toho, kdo zná klíč k1 (k2). MAC proto také zajišťuje službu **autentizace původu dat**. Protože je to symetrická technika, **nezaručuje nepopiratelnost**. Nezávislá třetí strana nemůže v případě sporu rozhodnout, zda data i s jejich zabezpečením vytvořil odesílatel nebo příjemce.

12. Hašovací funkce

Hašovací funkce jsou silným nástrojem moderní kryptologie. Jsou jednou z klíčových myšlenek druhé poloviny dvacátého století a přinesly řadu nových použití. V jejich základu jsou pojmy jednocestnosti a bezkoliznosti.

12.1. Definice

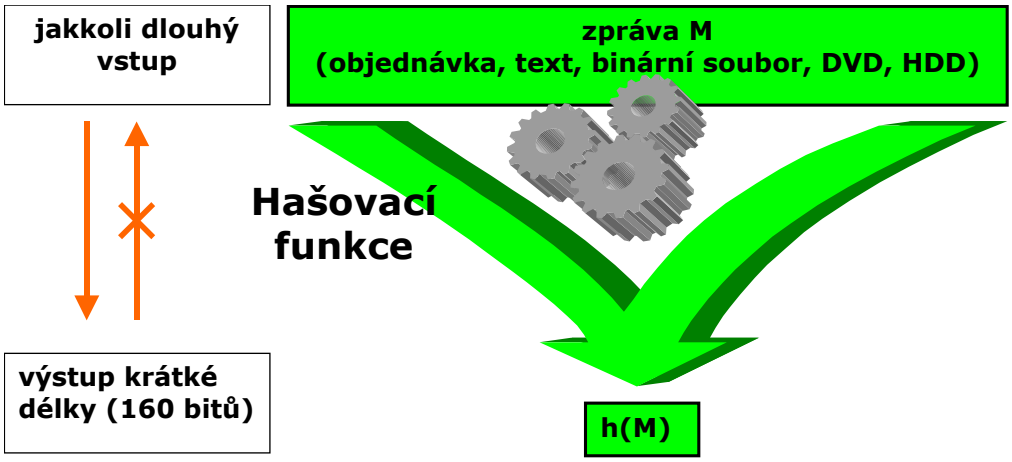
Definice: jednosměrná (jednocestná) funkce

Funkci $f: X \rightarrow Y$ nazveme jednosměrnou (jednocestnou), jestliže z jakékoli hodnoty $x \in X$ je snadné vypočítat $y = f(x)$, ale pro náhodně vybranou hodnotu $y \in f(X)$ je výpočetně nemožné (tj. současnými výpočetními prostředky neproveditelné) najít její vzor $x \in X$ tak, aby $y = f(x)$.

Definice: bezkolizní funkce

Funkci $f: X \rightarrow Y$ nazveme bezkolizní, jestliže je výpočetně nemožné nalézt různá $x, x' \in X$ tak, že $f(x) = f(x')$.

Definice: hašovací funkce
Mějme přirozená čísla L, n a nechť X je množina všech binárních řetězců délky 0 až L (prázdný řetězec je platným vstupem a má délku nula). Funkci h: $X \rightarrow \{0,1\}^n$ nazveme hašovací funkcí, jestliže je jednosměrná a bezkolizní. Říkáme, že každému binárnímu řetězci z množiny X přiřadí binární **hašový kód (hash, haš)** délky n bitů.



Obr.: Hašovací funkce

Poznámky:
V praxi bývá $L = 2^{64} - 1$, $L = 2^{128} - 1$ apod., takže vstupem hašovací funkce je libovolný a prakticky neomezeně dlouhý binární řetězec M. Výstupem je naopak hašový kód h(M) pevné, krátké a předem stanovené délky n bitů, obvykle stovky bitů.

12.2. Hašovací funkce jako náhodné orákulum
Orákulum nazýváme libovolný stroj (například program, losovací osudí apod.), který na základě vstupu odpovídá nějakým výstupem, přičemž na tentýž vstup zadaný podruhé odpovídá tímtéž výstupem. Náhodné orákulum pak na nový vstup odpovídá náhodným výběrem z množiny všech možných výstupů. Z hlediska bezpečnosti bychom byli rádi, kdyby se hašovací funkce chovala jako náhodné orákulum.

12.3. Odolnost proti nalezení vzoru
Základní vlastností hašovací funkce je její jednocestnost, tedy odolnost proti nalezení vzoru k danému hašovému kódu.

12.4. Odolnost proti nalezení druhého vzoru
Často se setkáváme s úlohou, aby pro danou zprávu x a její hašový kód $h(x)$ byla nalezena druhá zpráva y tak, aby měla tentýž hašový kód jako x. Pak by bylo možné k dané zprávě x a jejímu digitálnímu podpisu vyrobit padělanou zprávu y, která by měla stejnou haš (a tudíž stejný digitální podpis) jako zpráva existující. Proto se zavádí požadavek odolnosti proti tomuto útoku.
Řekneme, že hašovací funkce h je odolná proti nalezení druhého vzoru, jestliže pro *daný* náhodný vzor x je výpočetně nemožné nalézt druhý vzor $y \neq x$ tak, že $h(x) = h(y)$.

Pokud se bude hašovací funkce f: $X \rightarrow \{0,1\}^n$ chovat jako náhodné orákulum, bude složitost nalezení vzoru i druhého vzoru k danému hašovacímu kódu rovna 2^n.

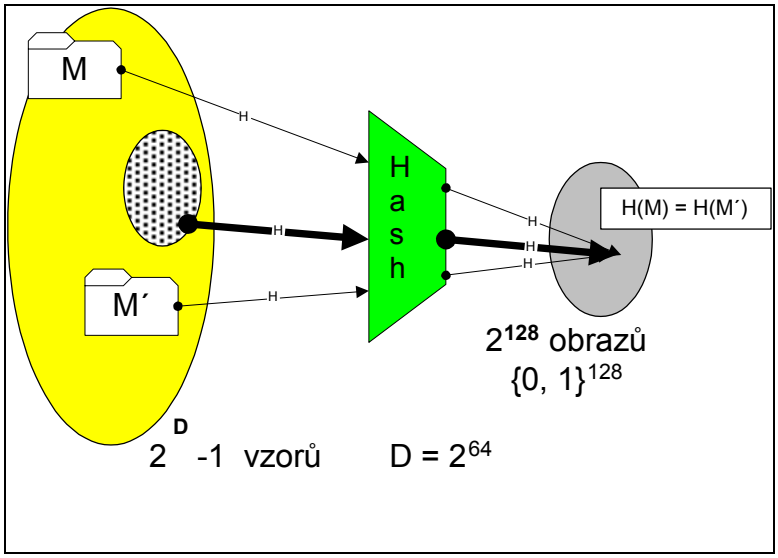
Je-li nalezena cesta, jak vzory nalézat jednodušeji, hovoříme o prolomení hašovací funkce.

12.5. Odolnost proti nalezení kolize

Pokud se hašovací funkce f: X → $\{0,1\}^n$ bude chovat jako náhodné orákulum, bude složitost nalezení kolize (tj. *libovolných* dvou zpráv se stejnou haší) rovna přibližně $2^{n/2}$. Tato složitost vyplývá z narozeninového paradoxu, viz dále. U náhodného orákula ani u hašovací funkce bychom neměli nalézt žádnou rychlejší metodu nalézání kolizí.

Pokud je nalezena cesta, jak kolize nalézat jednodušeji, hovoříme o prolomení hašovací funkce.

12.6. Častý omyl - nepochopení pojmu bezkoliznost



Obr. : Kolize

Pojem bezkoliznosti neznamená, že kolize u hašovací funkce neexistují, jen to, že je neumíme nalézt. Je zřejmé, že kolize musí nastávat, protože prostor možných zpráv X je mnohem větší než prostor možných hašovacích kódů. Například u MD5 možných zpráv o délce 0 až L je $1 + 2^1 + ... + 2^L = \mathbf{2^{L+1} - 1}$, kde $L = 2^{64} - 1$, zatímco hašovacích kódů je pouze 2^{128}. **Musí proto existovat ohromné množství zpráv, vedoucích na tentýž hašový kód** - zde na stejný hašový kód vede v průměru $2^{L+1-128}$ zpráv. Pointa je v tom, že nalezení byť jediné kolize je nad naše výpočetní možnosti i s využitím narozeninového paradoxu. Z principu definice hašovací funkce vyplývá existence neuvěřitelného množství kolizí, ale stejně tak z její definice vyplývá, že nalezení těchto kolizí musí být pro nás příliš těžká a výpočetně nemožná úloha.

12.7. Narozeninový paradox

Jestliže kolize zákonitě existují, položme si otázku, jak velká musí být množina náhodných zpráv, aby v ní existovaly s nemalou pravděpodobností dvě různé zprávy se stejnou haší, tj. aby nastala kolize. Tuto otázku řeší tzv. narozeninový paradox, od něhož se odvozuje bezpečnost hašovacích funkcí. Říká, kolik zpráv je nutno zhašovat, aby s cca 50% pravděpodobností nastala kolize v množině vzniklých hašovacích kódů. Například pro 160bitový hašový kód bychom očekávali $1/2 * 2^{160}$ zpráv, paradoxně je to pouhých 2^{80} zpráv.

Tvrzení: Narozeninový paradox. Mějme množinu M *m* různých prvků a proveďme výběr *k* prvků po jednom s vracením do množiny M. Potom pravděpodobnost, že se ve výběru objeví alespoň jeden prvek více než jednou, je $P(m, k) = 1 - (m(m-1) ... (m - k + 1)/ m^k)$. Pro $k = O(m^{1/2})$ a *m* jdoucí do nekonečna je $P(m, k)$ rovno přibližně $1 - \exp(-k^2/2m)$.

Důsledek: Pro m velké se ve výběru $k \approx m^{1/2}$ prvků z M s cca 50% pravděpodobností naleznou dva prvky shodné.

Důsledek pro kolize hašovacích funkcí. Pro hašovací funkce s n bitovým hašovým kódem máme postačí zhašovat $2^{n/2}$ zpráv, abychom s přibližně 50% pravděpodobností narozeninovým paradoxem nalezli kolizi.

Paradoxnost. Máme P(365, 23) = 0.507, P(365, 30)= 0.706. Pro čísla $m = 365$ a $k = 23$ interpretujeme tvrzení tak, že skupina náhodně vybraných 23 lidí postačí k tomu, aby se mezi nimi s cca 50% pravděpodobností našla dvojice, slavící narozeniny tentýž den. U skupiny 30 lidí je pravděpodobnost už 70%. Tvrzení se zdá paradoxní protože, ač je vyřčeno jinak, obvykle ho vnímáme ve smyslu "kolik lidí je potřeba, aby se k danému člověku našel jiný, slavící narozeniny ve stejný den". V této podbízející se interpretaci hledáme jedny konkrétní narozeniny, nikoli "jakékoliv shodné" narozeniny. Je to přesně rozdíl mezi hledáním kolize a hledáním druhého vzoru.

12.8. Konstrukce moderních hašovacích funkcí

U moderních hašovacích funkcí může být zpráva velmi dlouhá, například $L = 2^{64}$-1 bitů. Je zřejmé, že takovou zprávu musíme zpracovávat po částech, nikoli najednou. Také v komunikacích je přirozené, že zprávu dostáváme po částech a nemůžeme ji z paměťových důvodů ukládat celou pro jednorázové zhašování.

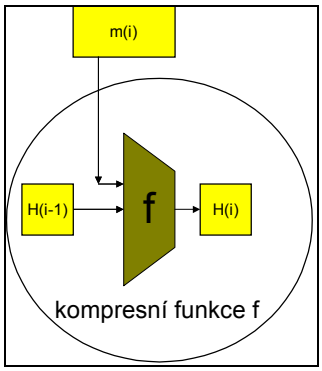
Odtud vyplývá nutnost **hašování zprávy sekvenčně po blocích**. Zpráva se proto zarovnává na celistvé bloky a za ní se také často doplňuje binární číslo, reprezentující její délku.

Předpokládejme tedy, že po doplnění máme hašovací funkci o n-bitovém hašovacím kódu a zprávu M zarovnanou na m-bitové bloky $m_1,..., m_N$.

Všechny současné prakticky používané hašovací funkce používají Damgard-Merklův (DM) princip iterativní hašovací funkce s využitím tzv. kompresní funkce.

Damgard-Merklova konstrukce

Kompresní funkce f zpracovává aktuální blok zprávy m_i. Výsledkem je určitá hodnota, které se říká kontext a označuje se H_i. Hodnota kontextu pak tvoří vstup do kompresní funkce v dalším kroku. Kompresní funkce má tedy dva vstupy - kontext a aktuální blok zprávy a jeden výstup, kterým je nový kontext.

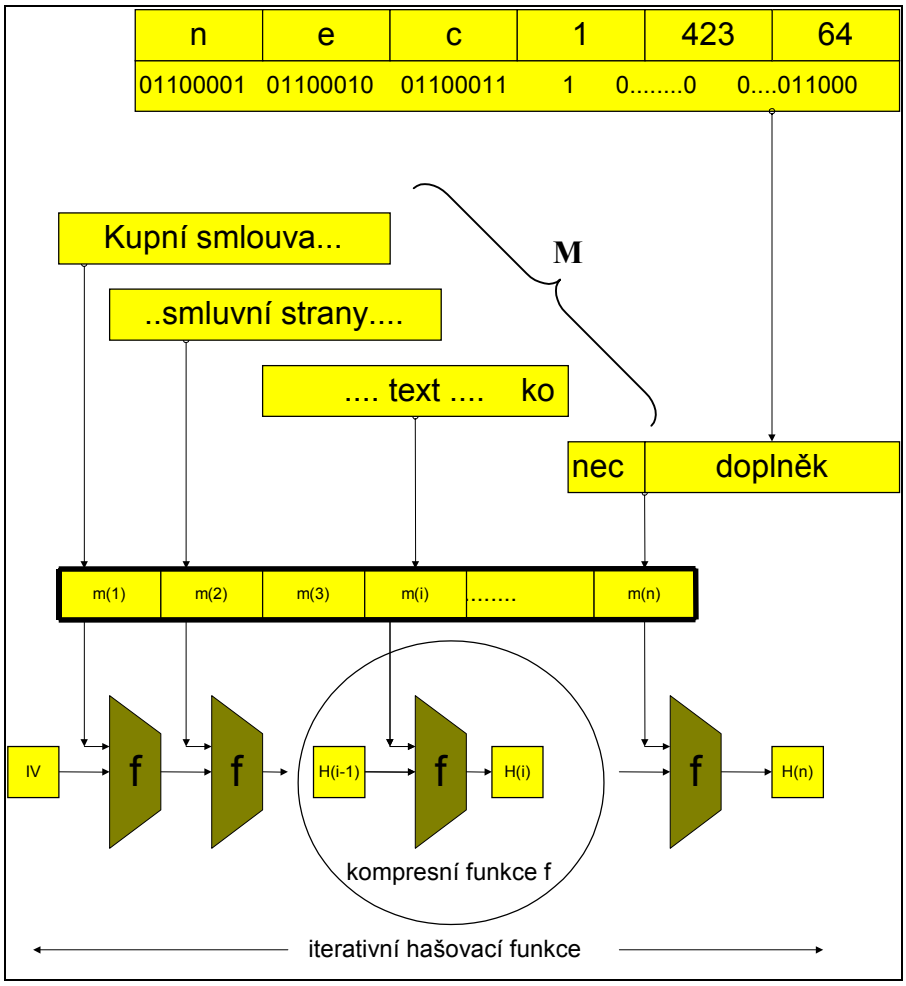


Odtud vznikla iterativní konstrukce, popsaná vzorcem

$H_i = f(H_{i-1}, m_i)$,

$H_0 = \text{IV}$,

která se nazývá Damgard-Merklova konstrukce (resp. Merklova meta metoda), kterou oba dva nezávisle navrhli na konferenci Crypto 1989.



Obr.: Doplňování, kompresní funkce a iterativní hašovací funkce

Hašování probíhá postupně po jednotlivých blocích m_i v cyklu podle *i* od 1 do *N*. Kompresní funkce *f* v i-tém kroku (i = 1,..., *N*) zpracuje vždy daný kontext H_{i-1} a blok zprávy m_i na nový kontext H_i. Šíře kontextu je většinou stejná jako šíře výstupního kódu, tj. n bitů.

Počáteční hodnota kontextu H_0 se nazývá inicializační hodnota (IV). Je definována jako nějaká konstanta v popisu každé iterativní hašovací funkce.

Vidíme, že název kompresní funkce je vhodný, neboť funkce *f* zpracovává širší vstup (H_{i-1}, m_i) na užší výstup H_i.

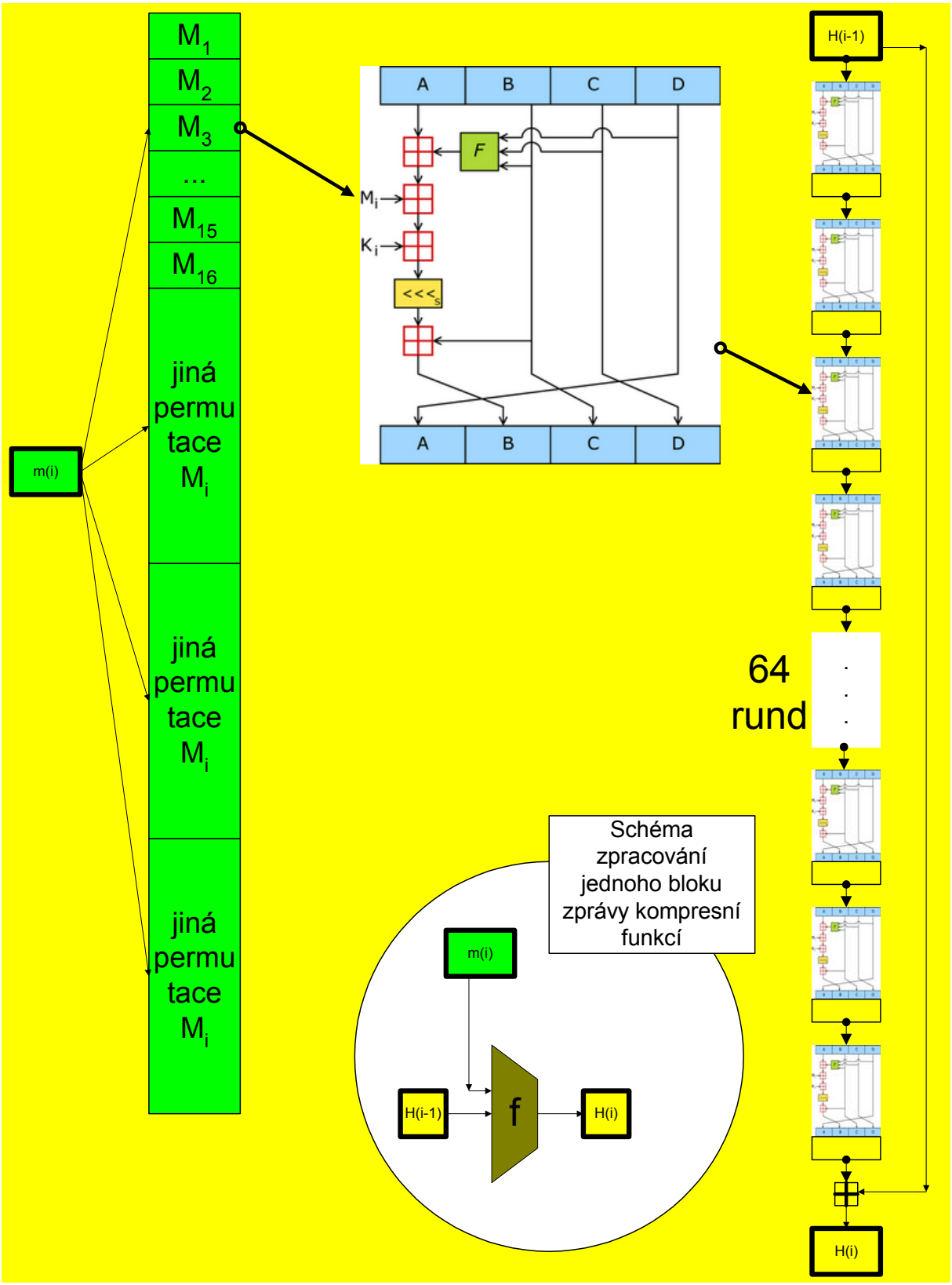
Po zhašování posledního bloku m_N dostáváme kontext H_N, z něhož bereme buď celou délku nebo část jako výslednou haš. U funkce MD5 je šířka kontextu 128 bitů a výslednou haš tvoří všech 128 bitů kontextu H_N.

12.9. Příklad hašovací funkce MD5

U MD5 tvoří kontext 128 bitů a e realizován čtyřmi 32bitovými slovy A, B, C a D. Na obrázku vidíme hašování jednoho bloku m(i) o 512 bitech v 64 krocích (rundách). Blok m(i) je rozdělen na 16 32bitových slov M_0, M_1, ..., M_{15}, a tato posloupnost je opakována 4x za sebou (v různých permutacích), což tvoří 64 vstupů. V každé rundě kompresní funkce se její kontext ovlivní vždy jedním ze slov M_j. Po 64 rundách dojde ještě k přičtení původního kontextu (H_{i-1}) k výsledku (tzv. Davies-Meyerova konstrukce) a tak vznikne nový kontext H_i. Pokud by zpráva M měla jen jeden blok, byl by kontext (A, B, C, D) celkovým výsledkem. Pokud ne, pokračuje se stejným způsobem v hašování druhého bloku zprávy jakoby s

inicializační hodnotou H_{i-1}. Po zpracování bloku m_N máme v registrech výslednou 128bitovou haš H_N.

Pozn.: V obrázku značí plus ve čtverečku součet modulo 2^{32} a výraz x <<< s označuje cyklický posun 32bitového slova x o s bitů doleva.



Obr.: MD5

12.10. Kryptografické využití hašovacích funkcí

12.10.1. Standardy a protokoly symetrické a asymetrické kryptografie

Hašovací funkce se využívají jak v symetrické, tak asymetrické kryptografii a jsou součástí moderních standardů a protokolů. Používají se například v SSL/TLS, PKCS, v autentizačních schématech, pro zabezpečení integrity dat, v digitálních podpisech apod.

12.10.2. Digitální otisk dat

Hašovací funkce vytváří z jakkoliv velkých dat de facto jejich identifikátor, neboť hašový kód díky bezkoliznosti "jednoznačně" identifikuje tato vstupní data. Nejsme totiž schopni nalézt jinou zprávu (jiná data), která by měla stejnou haš. Můžeme proto hovořit o tom, že jde o **digitální otisk dat** (**digital fingerprint**) nebo **výtah zprávy** (**message digest**). Jakákoli data lze tedy identifikovat digitálním otiskem mající řádově pár set bitů. V řadě zemí byly digitální otisky dat z hlediska identifikace dat legislativně postaveny de facto na roveň identifikace lidí pomocí otisků prstů. Je proto obvyklé, že v kryptosystémech digitálního podpisu se nepodepisují vlastní data, ale pouze jejich hašové kódy. To je výhodné, protože podepisované zprávy nebo soubory dat mohou být velmi dlouhé, zatímco na podpis krátkého hašového kódu postačí jedna podepisovací operace.

12.10.3. Kontrola integrity

Hašovací kódy se mohou díky výše uvedené vlastnosti použít přímo ke **kontrole integrity** přenášených zpráv podobně jako se využívají zabezpečovací kódy CRC (Cyclic Redundancy Check). Pokud se totiž zpráva naruší, změní se i její hašovací kód. Neporušenost zprávy můžeme proto kontrolovat neporušeností jejího hašovacího kódu.

12.10.4. Ukládání přihlašovacích hesel

Vlastnost jednocestnosti se využívá ke kontrole hesel v operačních systémech. Hesla se zde neukládají přímo, ale pouze jejich haše. Haše nijak neodhalují hesla, z nichž jsou vypočteny, ale přitom umožňují kontrolu jejich správnosti při přihlašování uživatelů do systému.

12.10.5. Pseudonáhodné funkce (PRF)

Pokud je hašovací funkce kvalitní, můžeme pomocí ní vytvářet pseudonáhodné funkce (PRF, pseudorandom function). Například standard PKCS#5 [15] umožňuje využít hašovací funkci k tvorbě "náhodného" šifrovacího klíče z passwordu pomocí pseudonáhodné funkce PRF jako klíč = PRF(password). Předpis PRF je vidět z obrázku a spočívá v mnohonásobném hašování passwordu a soli. Počet hašování je dán konstantou c. Tato PRF se například používá k tvorbě klíčů z passwordů.

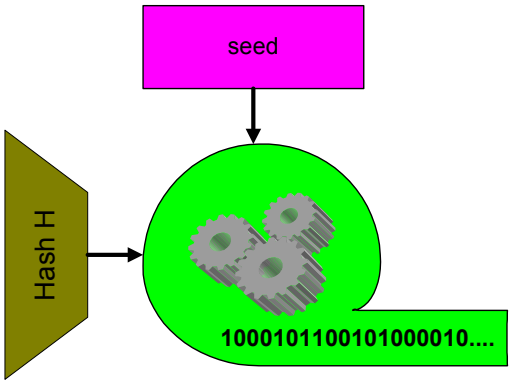
$T_1 = Hash\ (P \parallel S)$
$T_2 = Hash\ (T_1)$
…
$T_c = Hash\ (T_{c-1})$

šifrovací klíč $DK = T_c<0..dkLen-1>$

Obr.: Tvorba klíče DK z passwordu P a soli S podle PKCS#5

12.10.6. Pseudonáhodné generátory (PRNG)

Přímým použitím kvalitní hašovací funkce lze vytvářet pseudonáhodné generátory (PRNG, pseudorandom Number Generator). Obvykle vyjdeme z krátkého řetězce (náhodných) dat (seed) s dostatečnou entropií, který pomocí hašovací funkce expandujeme do posloupnosti pseudonáhodných čísel požadované délky.



Obr.: Hašovací funkce v konstrukci PRNG

Například standard PKCS#1 v.2.1 definuje pseudonáhodný generátor MGF1 (Mask Generation Function) pomocí hašovací funkce H s počátečním nastavením seed takto:

H(seed || 0x00000000), H(seed || 0x00000001), H(seed || 0x00000002), ...

Pokud se hašovací funkce chová jako náhodné orákulum, je tato posloupnost náhodná.

12.11. Klíčovaný hašový autentizační kód HMAC

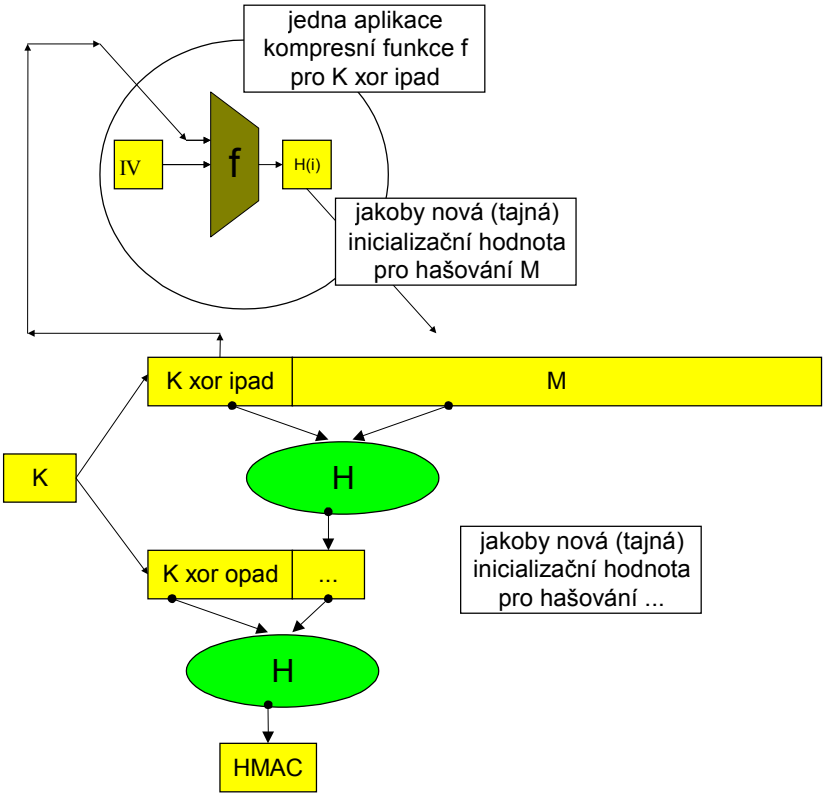
Klíčované hašové autentizační kódy zpráv HMAC zpracovávají hašováním nejen zprávu M, ale spolu s ní i nějaký tajný klíč K. Jsou proto podobné autentizačnímu kódu zprávy MAC, ale místo blokové šifry využívají hašovací funkci. Používají se jak k nepadělatelnému zabezpečení zpráv, tak k autentizaci (prokazováním znalosti tajného klíče K). Klíčovaný hašový autentizační kód je obecná konstrukce, která využívá nějakou hašovací funkci. Podle toho, jakou používá konkrétně, označuje se výsledek, například HMAC-SHA-1(M, K) používá SHA-1, kde M je zpráva a K je tajný klíč.

12.11.1. Obecná definice HMAC

HMAC je definován ve standardu FIPS PUB 198 [4]. Klíč K je zde modifikován konstantami *ipad* a *opad* a HMAC definujeme jako

HMAC-H(M, K) = H((K xor *opad*) || H((K xor *ipad*)|| M)),

kde || označuje zřetězení. Situaci znázorňuje obrázek.



HMAC-H(K, M) = H((K xor opad) || H((K xor ipad)|| M))

Obr.: Klíčovaný hašový autentizační kód zprávy HMAC-H(K, M)

12.11.2. Nepadělatelný integritní kód, autentizace původu dat

Zabezpečovací kód HMAC(M, K) připojený za zprávu M detekuje neúmyslnou chybu při jejím přenosu. Případnému útočníkovi také zabraňuje změnit zprávu a současně změnit HMAC, protože bez znalosti klíče K nelze nový HMAC vypočítat. HMAC může být proto chápán jako nepadělatelný integritní kód, který samotná haš neposkytuje. Pro komunikujícího partnera je správný HMAC také autentizací původu dat, protože odesílatel musel znát hodnotu tajného klíče K.

12.11.3. Průkaz znalosti při autentizaci entit

HMAC může být použit také jako průkaz znalosti tajného sdíleného tajemství (K) při autentizaci entit. Princip průkazu je tento. Dotazovatel odešle nějakou náhodnou výzvu (řetězec) *challenge* a od prokazovatele obdrží odpověď *response* = HMAC(*challenge*, K). Nyní ví, že prokazovatel zná hodnotu tajného klíče K. Přitom případný útočník na komunikačním kanálu z hodnoty *response* klíč K nemůže odvodit.

-oOo-