Základy moderní kryptologie - Symetrická kryptografie III.

operační mody blokových šifer a hašovací funkce

Vlastimil Klíma

verze: 1.4, 20. 4. 2005

Abstrakt

Cílem třech přednášek (Symetrická kryptografie I, II a III) je

a) ukázat, že moderní kryptologie se zabývá mnohem širším okruhem věcí než jen utajováním zpráv a jejich luštěním,
b) seznámit s některými novými myšlenkami,
c) a věnovat se více jedné části moderní kryptologie, tzv. symetrickým schématům.

Vzhledem k rozsahu těchto přednášek, které mají úvodní přehledový charakter, nebude možné postihnout ani klíčové, ani nejkrásnější myšlenky této vědy, ale jen některé nejpoužívanější. Následující texty vychází částečně z citované a doporučené literatury, jsou však nutně zatíženy subjektivním výkladem.

Doporučená literatura:

*Základní příručka **on-line**:*
Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996, dostupné celé on-line na
http://www.cacr.math.uwaterloo.ca/hac/

Často doporučovaná alternativa:
Doug Stinson, *Cryptography: Theory and Practice*, CRC Press, 1995

Historie:
David Kahn, *The Codebreakers*, Scribner, 1996

Internetový portál:
http://theory.lcs.mit.edu/~rivest/crypto-security.html

Kontakt a osobní stránky:
v.klima@volny.cz, http://cryptography.hyperlink.cz

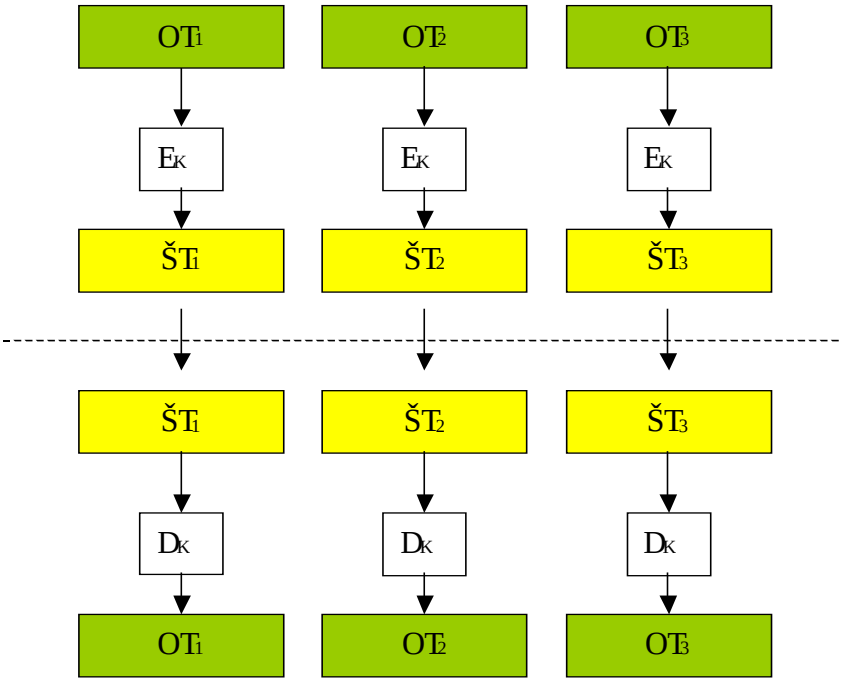
Obsah

11. Operační mody blokových šifer

Operační mody blokových šifer jsou způsoby použití blokových šifer v daném kryptosystému. kde otevřeným textem není jen jeden blok blokové šifry, ale obecně posloupnost znaků dané abecedy. U moderních blokových šifer chápeme jako znaky bajty, i když se délka bloku *N* uvádí v bitech. Obvykle *N* = 64 nebo 128. 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. ECB (Electronic Codebook)

Tento operační modus se nazývá elektronická kódová kniha a je základním modem. Posloupnost bloků otevřeného textu OT_1, OT_2,... OT_n se šifruje tak, že každý blok je šifrován zvlášť, což lze vyjádřit vztahem $ŠT_i = E_K(OT_i)$, i = 1, 2, ..., n.



Obr.: Modus ECB

11.1.1. Informace, vyzařující ze šifrového textu v modu ECB

Nevýhodou takového typu šifrování je, že **stejné bloky otevřeného textu mají vždy stejný šifrový obraz**. Pokud nalezneme několik shodných bloků šifrového textu, že může to v určitém kontextu dokonce rozkrývat i hodnotu otevřeného bloku (například prázdné sektory na disku jsou vyplněny hodnotou 0xFF apod.

11.1.2. Integrita a modus ECB

Ve zprávě, šifrované modem ECB **útočník může 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. CBC (Cipher Block Chaining)

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

Synchronní proudové šifry mají nedostatečnou vlastnost difúze neboť pracují jen nad jednotlivými znaky abecedy. Moderní blokové šifry naproti tomu dosahují velmi dobrých vlastností jak difúze, tak konfúze.

Je tomu tak proto, že vznikaly už v době elektronických šifrátorů, kdy nebyl problém vytvářet iterativní součinové šifry. Větší počet iterací a vhodné rundovní funkce pak mohly vlastnost difúze a konfúze zajistit s dostatečnou rezervou. Provést tolik operací v případě ručního šifrování, u mechanických nebo elektromechanických šifrátorů nebylo technicky možné. Difúze a konfúze při zpracování jednoho bloku otevřeného textu u moderních blokových šifer byla tedy dosažena. Je to ale málo, neboť otevřeným textem je obvykle velmi mnoho bloků.

Aby blokové šifry rozšířily difúzi na více bloků, byl definován modus řetězení šifrového textu (CBC). Každý blok otevřeného textu se v něm nejprve modifikuje předchozím blokem šifrového textu, a teprve poté se šifruje. To zajišťuje, že běžný šifrový blok závisí na celém předchozím otevřeném textu z důvodu řetězení této závislosti přes předchozí šifrový text.

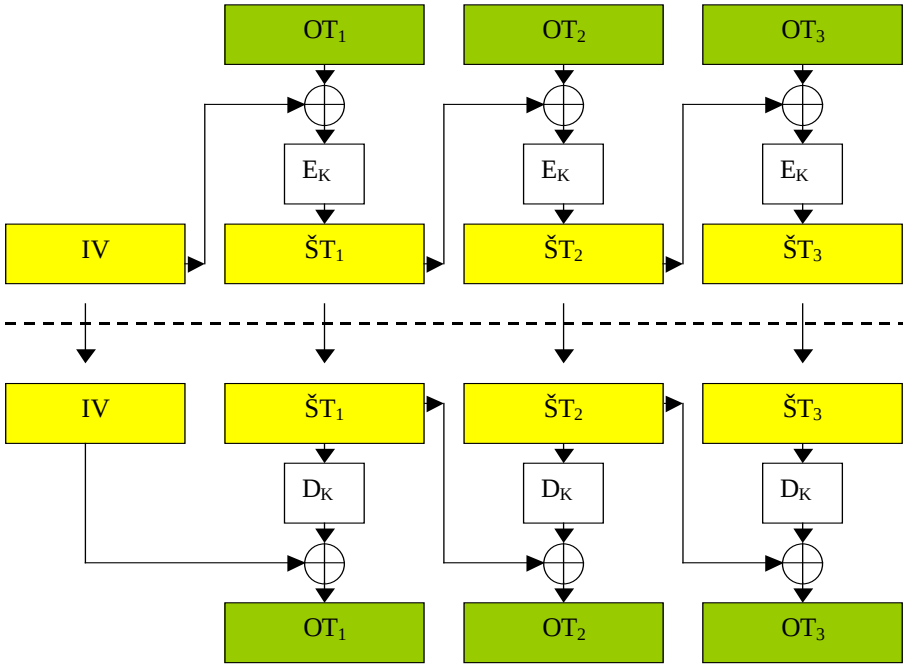
11.2.2. Definice CBC

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

$\check{S}T_0 = IV,$
$\check{S}T_i = E_K(OT_i \oplus \check{S}T_{i-1}), i = 1, 2, ..., n.$

a odšifrování probíhá podle vztahů:
$\check{S}T_0 = IV,$
$OT_i = \check{S}T_{i-1} \oplus D_K(\check{S}T_i), i = 1, 2, ..., n.$

Náhodný inicializační vektor způsobí, že pokud bychom dvakrát šifrovali shodný první blok OT_1, různé náhodné IV ho modifikují na různé náhodné vstupní bloky. Výsledný blok šifrového textu $\check{S}T_1$ 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. **Budeme-li šifrovat jeden a tentýž, byť i velmi dlouhý, otevřený soubor dat dvakrát, obdržíme naprosto odlišný šifrový text.**



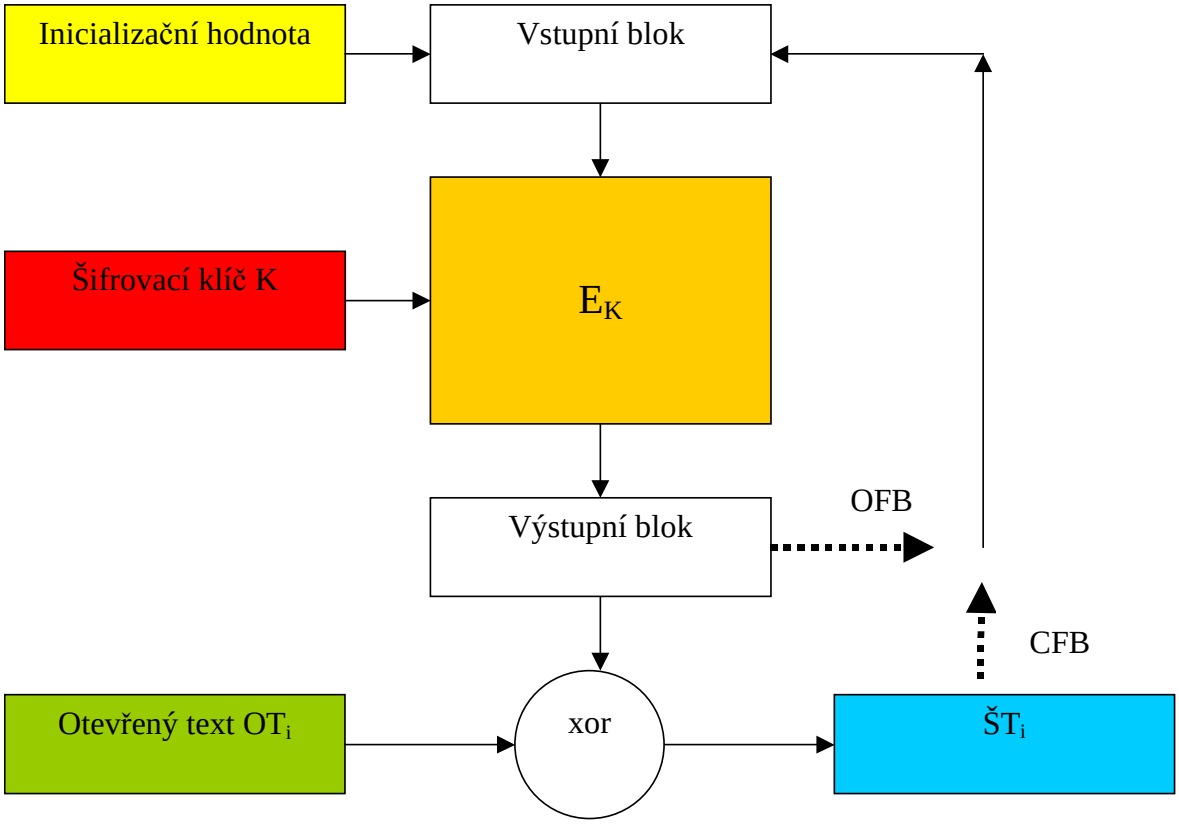
Obr.: Modus CBC

11.2.3. Samosynchronizace modu CBC

Řetězení mírně znesnadňuje útoky na modus ECB popsané výše. Z definice modu CBC však vyplývá tzv. vlastnost samosynchronizace. 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 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 CFB a OFB (Cipher Feedback, Output Feedback)

Tyto operační mody převádí blokovou šifru na proudovou. 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) nebo vzniklý šifrový text (v modu CFB) jsou vedeny na vstup blokové šifry a jejich 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.



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

Předpis pro zašifrování v modu CFB:
$ŠT_0 = IV$,
$ŠT_i = OT_i \oplus E_K(ŠT_{i-1})$, i = 1, 2, ..., n.
Předpis pro odšifrování v modu CFB:
$ŠT_0 = IV$,
$OT_i = ŠT_i \oplus E_K(ŠT_{i-1})$, i = 1, 2, ..., n.

Předpis pro zašifrování v modu OFB:
$H = IV = ŠT_0$,
pro i = 1, 2, ..., n: $\{H = E_K(H), ŠT_i = OT_i \oplus H\}$
Předpis pro odšifrování v modu OFB:
$H = IV = ŠT_0$,
pro i = 1, 2, ..., n: $\{H = E_K(H), OT_i = ŠT_i \oplus H\}$

Všechny uvedené operační mody definují normy FIPS PUB 81, ANSI X3.106, ISO 8732 a ISO/IEC 10116. Povšimněte si, že **v modech OFB a CFB se bloková šifra používá jen jednosměrně** tj. jen transformace E_K. Transformaci D_K není použita. To je výhodné při hardwarové realizaci.

11.3.1. Samosynchronizace, neúplná zpětná vazba a délka periody

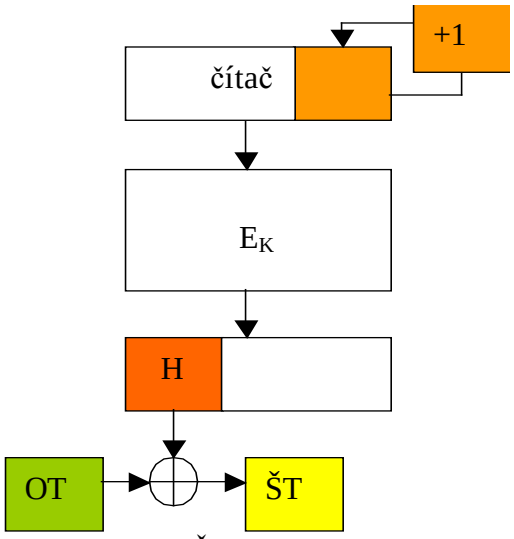
Poznamenejme, že z výstupu nemusíme použít celý blok hesla, ale jen jeho část, například *b* bitů. V tom případě se *b* bitů hesla (u OFB) nebo *b* bitů vzniklého šifrového textu (u CFB) vede zprava do vstupního registru, přičemž původní obsah vstupního registru se posune doleva o *b* bitů (*b* bitů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

Modus CFB má vlastnost samosynchronizace, a to podle délky zpětné vazby. Je-li *b* bitů, pak postačí dva nenarušené *b*-bitové bloky šifrového textu, 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, 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 nebo nikoli. Pro 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

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.



Obr.: Čítačový modus

Používá také 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 tzv. dvojímu použití hesla a mohlo by dojít k rozluštění otevřeného textu dotčených zpráv. Podrobnější definice s příklady vytváření čítače jsou uvedeny v [25].

Předpis pro zašifrování v modu CTR:
$CTR_i = (IV + i - 1) \bmod 2^B$, $H_i = E_K(CTR_i)$, $ŠT_i = OT_i \oplus H_i$, i = 1,2,...
Předpis pro odšifrování v modu CTR:
$CTR_i = (IV + i - 1) \bmod 2^B$, $H_i = E_K(CTR_i)$, $OT_i = ŠT_i \oplus H_i$, i = 1,2,...

I když se jedná o nedávno standardizovaný modus, jeho myšlenku popsali v roce 1979 Diffie a Hellman [26]. Jedná se o čistou (synchronní) proudovou šifru, používající pouze

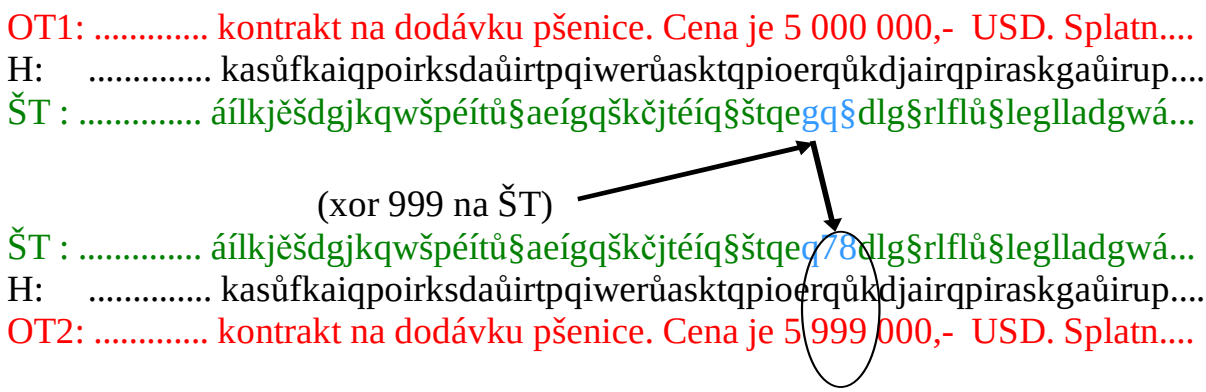
transformaci E_K i při dešifrování. Výstupní blok lze použít celý nebo jen jeho část. Smyslem modu je zaručit maximální periodu hesla, což je zaručeno periodou čítače.
Další výhodou je, že heslo může být vypočítáno jen na základě pozice otevřeného textu a IV, nezávisle na ničem jiném. Tuto vlastnost 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.

11.5. Metoda solení

Poznamenejme, že u operačních modů CBC, OFB, CFB i CTR je též možné využívat metodu solení IV. Spočívá v tom, že komunikujícímu protějšku se sice předává hodnota IV, ale k šifrování se použije jiná hodnota IV´ ("osolený IV"), která se na obou stranách vypočítá z IV a klíče K nějakým definovaným způsobem. Například to může být hašovací hodnota, vypočítaná ze zřetězení obou hodnot. Bezpečnostní výhodou je, že skutečně použitá inicializační hodnota IV´ se nikde neobjevuje na komunikačním kanálu. Metodou solení (salting) se zabývá norma PKCS#5 [27].

11.6. 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. Toho lze využít k různým útokům, princip ilustruje obrázek (poznamenejme, že šifrový text je smyšlený).



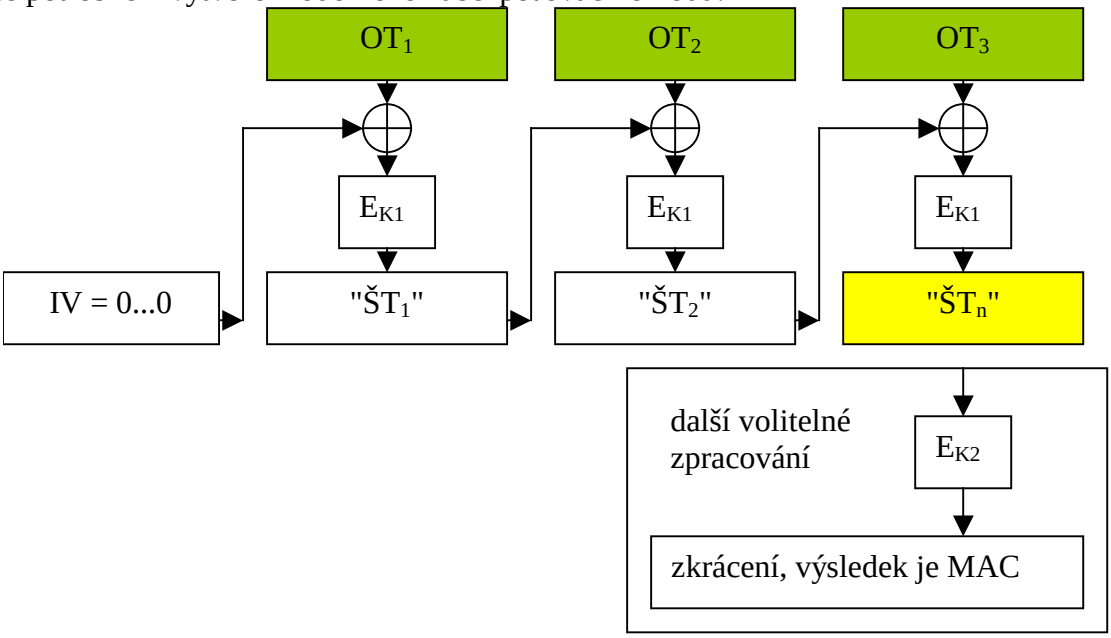
dešifrováním změněného šifrového textu (ŠT1 ⊕ d) obdržíme
(ŠT1 ⊕ d) ⊕ H = (OT1 ⊕ H ⊕ d) ⊕ H = OT1 ⊕ d

Obr.: Příklad útoku na proudovou šifru změnou šifrového textu

11.7. MAC (Message Authentication Code)

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) 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 (často polovina bloku) 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 (ev. přídavný 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.

11.8. Další mody

Přestože se zdá, že operačních modů je dost, není tomu tak. Nové potřeby vyžadují nové mody, příkladem budiž nutnost šifrovat data a současně počítat jejich zabezpečovací kód MAC. Současná kombinace modů CBC a MAC může však být ve výpočetně omezených zařízeních pomalá nebo příliš náročná na systémové prostředky (zejména paměť). Proto NIST vyhlásil novou iniciativu na vytváření nových modů. Blíže viz web NIST [25]. Například v květnu 2004 byl publikován nový **modus CCM pro šifrování dat a současně jejich autentizaci** [28]. Používá AES v čítačovém modu k šifrování v kombinaci s technikou CBC-MAC pro autentizaci. Práce na modech nejsou uzavřeny.

12. Hašovací funkce

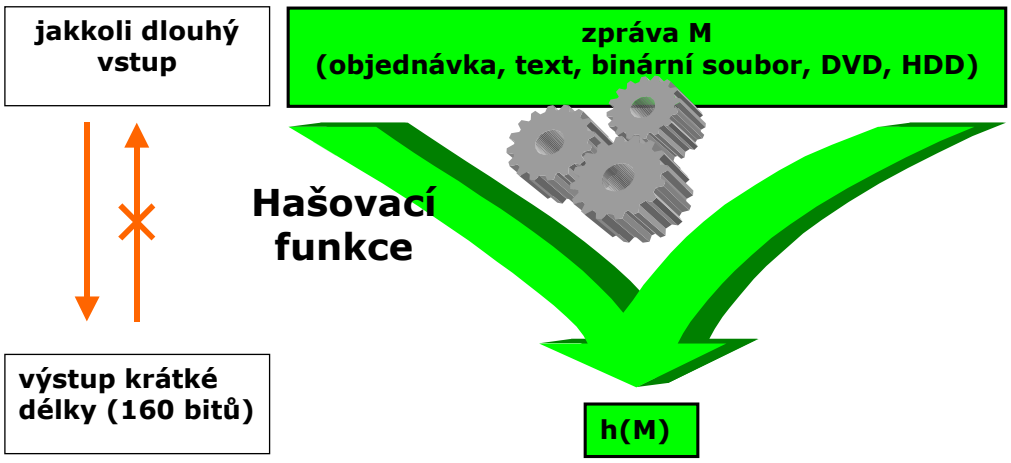
Hašovací funkce jsou silným nástrojem moderní kryptologie. Jsou jednou z klíčových myšlenek druhé poloviny dvacátého století a přinesly řadu nových použití. V jejich základu jsou pojmy jednocestnosti a bezkoliznosti.

12.1. Definice

Definice: jednosměrná (jednocestná) funkce
Funkci f: X → Y nazveme jednosměrnou (jednocestnou), jestliže z jakékoli hodnoty x ∈ X je snadné vypočítat y = f(x), ale pro náhodně vybranou hodnotu y ∈ f(X) neumíme (je výpočetně nemožné) najít její vzor x ∈ X tak, aby y = f(x).

Definice: bezkolizní funkce
Funkci f: X → Y nazveme bezkolizní, jestliže je výpočetně nemožné nalézt různá x, x´∈ X tak, že f(x) = f(x´).

Definice: hašovací funkce
Mějme přirozená čísla D, n a nechť X je množina všech binárních řetězců délky 0 až D (prázdný řetězec je platným vstupem a má délku nula). Funkci h: X → $\{0,1\}^n$ nazveme hašovací, 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 D = 2^{64} -1, D = 2^{128} - 1 apod., takže vstupem hašovací funkce je (z hlediska současných výpočetních aplikací) libovolný a prakticky neomezeně dlouhý binární řetězec M. Výstupem je naopak hašový kód h(M) pevné, krátké a předem stanovené délky n bitů.
Z vlastnosti jednocestnosti vyplývá, že pro dané M je jednoduché vypočítat hašový kód h(M), ale obráceně to není výpočetně zvládnutelné.

12.2. Hašovací funkce jako náhodné orákulum

Orákulum nazýváme libovolný stroj (stroj "podivuhodných vlastností"), který na základě vstupu odpovídá nějakým výstupem. Má pouze vlastnost, že na tentýž vstup odpovídá tímtéž výstupem. Náhodné orákulum je orákulum, které na nový vstup odpovídá náhodným výběrem výstupu z množiny možných výstupů.

Hašovací funkce jako náhodné orákulum
Z hlediska bezpečnosti bychom byli rádi, kdyby se hašovací funkce chovala jako náhodné orákulum. Odtud se odvozují bezpečnostní vlastnosti.

12.3. Odolnost proti nalezení druhého vzoru

Z vlastnosti jednocestnosti plyne odolnost hašovací funkce proti nalezení *druhého* vzoru neboli bezkoliznost druhého řádu.

Řekneme, že hašovací funkce *h* je odolná proti nalezení druhého vzoru, jestliže pro *daný* náhodný vzor *x* je výpočetně nezvládnutelné nalézt druhý vzor $y \neq x$ tak, že h(x) = h(y).

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

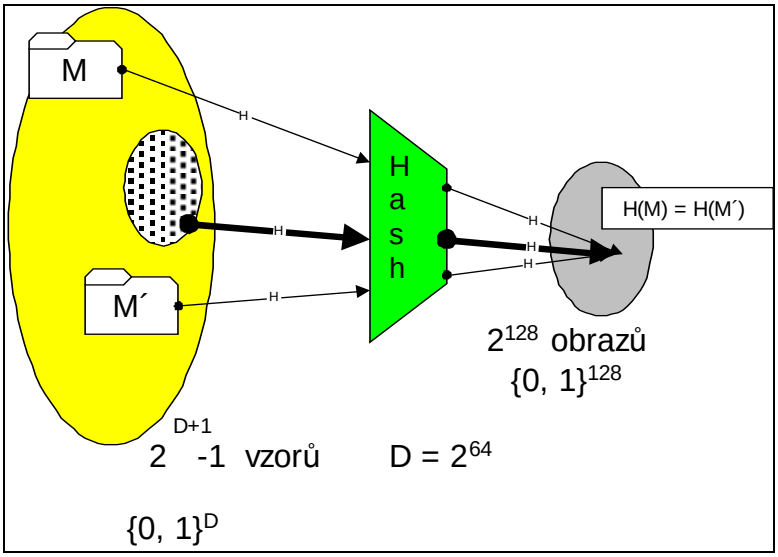
Je-li nalezena cesta, jak vzory nalézat jednodušeji, hovoříme o prolomení hašovací funkce.

12.4. Odolnost proti nalezení kolize

Pokud se hašovací funkce f: $\{0,1\}^D \rightarrow \{0,1\}^n$ bude chovat jako náhodné orákulum, bude složitost nalezení kolize rovna přibližně $2^{n/2}$. Tato složitost nalezení kolize $2^{n/2}$ vyplývá z tzv. narozeninového paradoxu, viz dále.U náhodného orákula neznáme rychlejší cestu nalézání kolizí.

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

12.5. Častý omyl - nepochopení pojmu bezkoliznost



Obr. : Kolize

Vezměme si příklad hašovací funkce MD5. Možných zpráv je mnoho ($1 + 2^1 + ... + 2^D = 2^{D+1} -1$), kde $D = 2^{64} -1$, a hašovacích kódů málo. U MD5 je jich pouze 2^{128}. Musí proto existovat ohromné množství zpráv, vedoucích na tentýž hašový kód - v průměru je to řádově 2^{D-127}. **Kolizí tedy existuje ohromné množství**. Pointa je v tom, že nalezení byť jediné kolize je nad naše výpočetní možnosti i s využitím narozeninového paradoxu. Z principu definice hašovací funkce tak vyplývá existence neuvěřitelného množství kolizí, ale stejně tak z její definice vyplývá, že nalezení těchto kolizí je pro nás příliš těžká a výpočetně neproveditelná úloha.

12.6. 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. Pro 160bitový hašový kód bychom očekávali $1/2 * 2^{160}$ zpráv, paradoxně je to pouhých 2^{80} zpráv.

Tvrzení: Mějme množinu M *n* různých koulí a proveďme výběr *k* koulí po jedné s vracením do množiny M. Potom pravděpodobnost, že se ve výběru objeví alespoň jedna koule více než jednou, je $P(n, k) = 1 - (n(n-1) \dots (n - k + 1)/ n^k)$. Pro $k = O(n^{1/2})$ a *n* jdoucí do nekonečna je $P(n, k)$ rovno přibližně $1 - \exp(-k^2/2n)$.

Důsledek: Pro *n* velké se ve výběru $k = (2n\ln 2)^{1/2}$ prvků z M s cca 50% pravděpodobností naleznou dva prvky shodné.

Důsledek pro kolize hašovacích funkcí. Obecně pro hašovací funkce s *m* bitovým hašovým kódem postačí zhašovat $2^{m/2}$ zpráv, abychom s přibližně 50% pravděpodobností nalezli kolizi.

Paradoxnost. Máme P(365, 23) = 0.507, P(365, 30)= 0.706. Pro čísla n = 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.

13. Konstrukce moderních hašovacích funkcí

U moderních hašovacích funkcí může být zpráva velmi dlouhá, například $D = 2^{64}$-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 po blocích a sekvenční způsob**.
Ze zpracování po blocích plyne také nutnost **zarovnání vstupní zprávy na celistvý počet bloků** před hašováním.

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

13.1. Zarovnání a doplnění o délku zprávy

Zarovnání musí být takové, aby umožňovalo jednoznačné odejmutí, jinak by vznikaly jednoduché kolize. Například při doplnění zprávy nulovými bity by nešlo rozeznat, kolik jich bylo doplněno, a zda některé z nich nejsou platnými bity zprávy, pokud by zpráva nulovými bity končila.
Například doplněná zpráva
10111100110101000000000000000000000000000000
by byla doplněním zpráv
10111100110101
101111001101010
1011110011010100
10111100110101000
...
10111100110101000000000000000000000000000000,
a všechny by proto vedly ke kolizi (měly by stejný hašový kód), dokonce multiplikativní kolizi.
Zarovnání u moderních hašovacích funkcí se definuje jako doplnění bitem 1 a poté potřebným počtem bitů 0. To umožňuje jednoznačné odejmutí doplňku.

Za potřebným počtem bitů je nutné doplnit délku původní zprávy (tzv. Damgard-Merklovo zesílení), jinak by bylo možné vést útok nalezením druhého vzoru zprávy. Doplnění pro blok délky m = 512, který je použit u hašovacích funkcí MD4, MD5, SHA-0, SHA-1, SHA-256 se provádí tak, že *M* je nejprve doplněna bitem 1, poté co nejmenším počtem nulových bitů (může jich být 0 - 447) tak, aby do celistvého násobku 512 bitů zbývalo ještě 64 bitů, a nakonec je těchto 64 bitů vyplněno 64bitovým vyjádřením počtu bitů původní zprávy *M*. Délka zprávy tedy také vstupuje do hašovacího procesu. Rezervace 64 bitů na délku zprávy umožňuje hašovat zprávy až do délky $D = 2^{64}$-1 bitů.

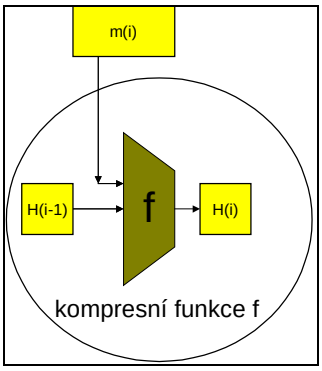
13.2. Damgard-Merklův iterativní princip moderních hašovacích funkcí

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

Damgard-Merklova konstrukce

Kompresní funkce f zpracovává aktuální blok zprávy m_i. **Výsledkem je určitá hodnota, které se říká kontext a označuje H_i.** Hodnota kontextu pak nutně tvoří vstup do

kompresní funkce v dalším kroku. Tím dostáváme požadavek, aby kompresní funkce měla dva vstupy - kontext a aktuální blok zprávy a jeden výstup - nový kontext.

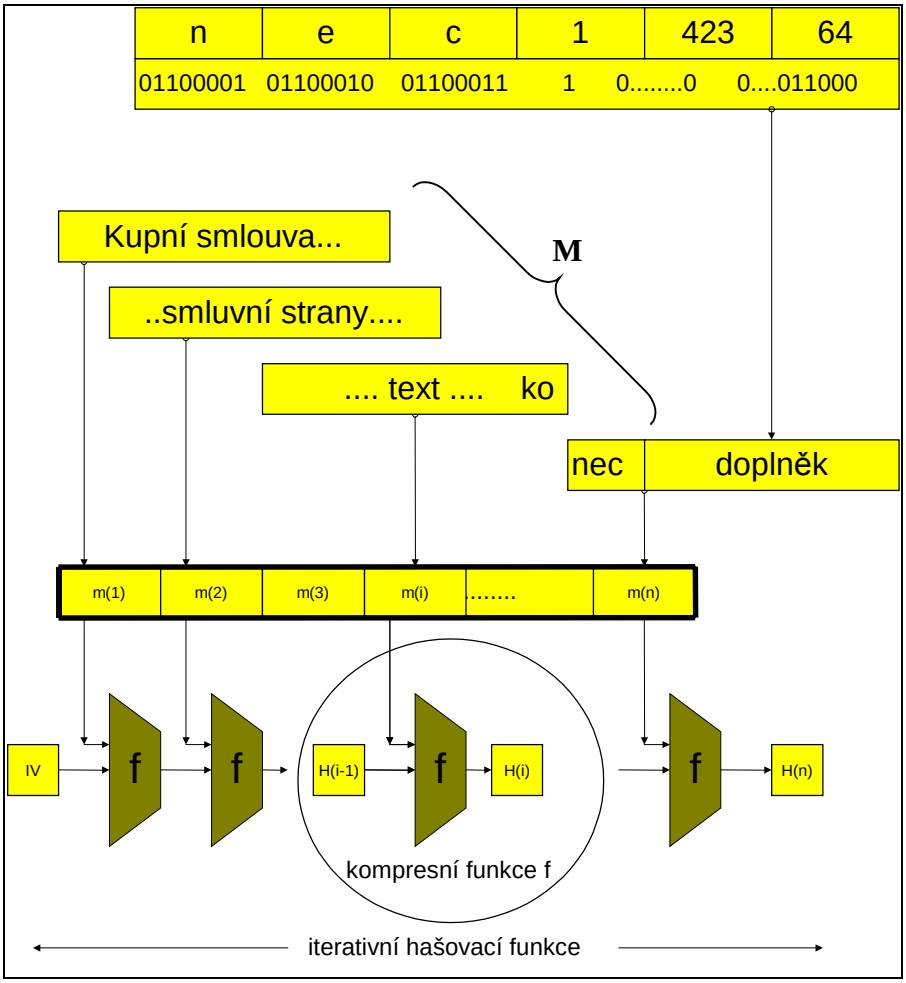


Odtud vznikla iterativní konstrukce, popsaná vzorcem

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

$H_0 = \mathrm{IV}$,

která se nazývá **Damgard-Merklova konstrukce** (resp. Merklova meta metoda), neboť ji 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.

Počáteční hodnota kontextu H_0 se nazývá inicializační hodnota (IV). Je dodefinována jako konstanta daná 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 mnohem kratší H_i, tedy blok zprávy m_i se sice funkčně promítne do H_i, ale současně dochází ke ztrátě informace (šířka kontextu H_0, H_1,...H_i,... zůstává stále stejná). Kontextem bývá obvykle několik 16bitových nebo 32bitových slov, u MD5 jsou to čtyři slova A, B, C, D (dohromady 128 bitů).

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.

13.3. Konstrukce kompresní funkce

Kompresní funkce musí být velmi robustní, aby zajistila dokonalé promíchání bitů zprávy a jednocestnost.

Jak tyto funkce konstruovat? Protože hašovací funkce musí být jednocestné a chovat se co nejvíce jako **náhodné orákulum**, máme možnost je konstruovat na bázi známých jednocestných funkcí. Můžeme použít **znalostí z oblasti blokových šifer**. Kvalitní bloková šifra $E_k(x)$ se při pevném klíči *k* také má chovat jako náhodné orákulum. Dále zaručuje, že známe-li jakoukoli množinu vstupů-výstupů, tj. otevřených-šifrových textů (x, y), nemůžeme odtud určit (díky složitosti) klíč *k*. **Vzhledem ke klíči je tak bloková šifra jednocestná**. Přesněji pro každé x je funkce $k \rightarrow E_k(x)$ jednocestná. Odtud vyplývá možnost konstrukce kompresní funkce takto:

$$\mathbf{H_i = E_{mi}(H_{i-1}),}$$

kde E je kvalitní bloková šifra.

Vezměme hašování krátké zprávy, která i se zarovnáním tvoří jediný blok zprávy. Máme $H_1 = E_{m1}(IV)$. Je vidět, že z hodnoty hašového kódu H_1 nejsme schopni určit m_1, čili máme zaručenu vlastnost jednocestnosti.

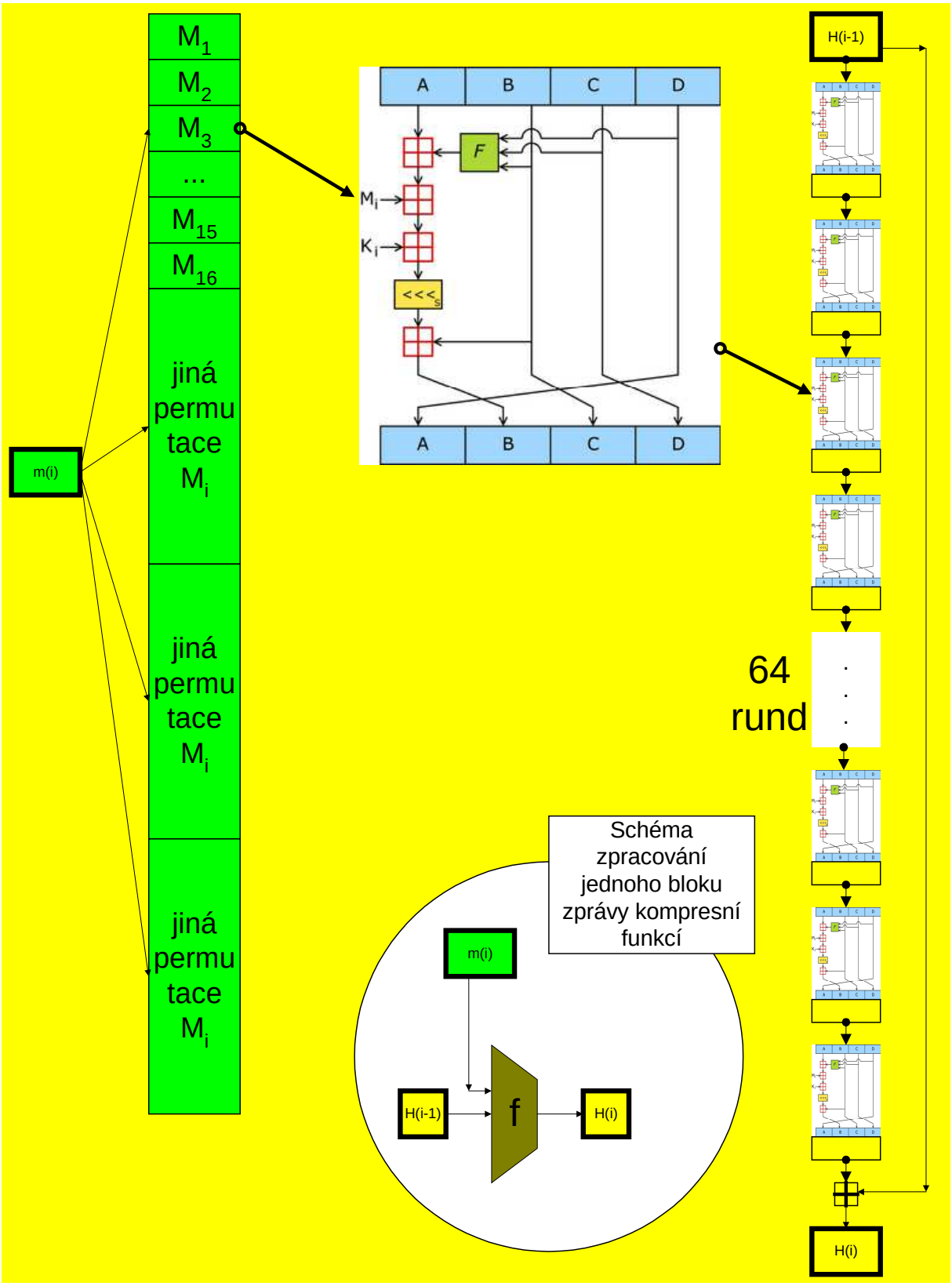
U moderních funkcí se používá navíc ještě jedna operace. Jedná se o tzv. Davies-Meyerovu konstrukci kompresní funkce, která zesiluje vlastnost jednocestnosti ještě přičtením vzoru před výstupem: $H_i = f(H_{i-1}, m_i) = E_{mi}(H_{i-1})$ xor H_{i-1}. Výstup je zde tedy navíc maskován vstupem, což ještě více ztěžuje případný zpětný chod.

"rundovní klíče" a příprava zprávy

Protože blok zprávy m_i je obvykle velmi velký (512 bitů) a klíče blokových šifer tak dlouhé nebývají, aplikuje se bloková šifra několikrát za sebou (v rundách), přičemž (rundovní) klíč postupně čerpá z bloku m_i. Na schématu MD5 (viz obr. dále) vidíme jak blokovou šifru, tak způsob, jakým je blok zprávy m_i v 16 rundách po částech používán jako rundovní klíč. Navíc se tento stavební blok ještě čtyřikrát opakuje, takže se každá část bloku m_i projeví na místě klíče dokonce čtyřikrát. Této fázi se proto někdy paradoxně říká příprava klíče, i když se jedná o přípravu zprávy.

14. Kompresní a hašovací funkce MD5

U MD5 tvoří kontext 4 32bitová slova A, B, C a D. Na obrázku vidíme zvětšenu jednu rundu hašování. m_i je jeden 512 bitový blok zprávy. Ten je rozdělen na 16 32bitových slov M_0, M_1, ..., M_{15}, a tato posloupnost je opakována 4x za sebou (v různých permutacích). Na obrázku vidíme, že v kompresní funkci se kontext "zašifruje" vždy jedním 32bitovým slovem M_i. Poznamenejme, že na místě dílčí funkce F v obrázku se po 16 rundách střídají 4 různé (nelineární i lineární) funkce (F, G, H, I) a v každé rundě se využívá jiná konstanta K_i. Po 64 rundách dojde ještě k přičtení původního kontextu (H_{i-1}) k výsledku podle Davies-Meyerovy konstrukce (xor je nahrazen aritmetickým součtem modulo 2^{32}). 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 m_2 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 modulární součet a výraz x <<< s označuje cyklický posun 32bitového slova x o s bitů doleva.


M1
M2
M3
...
M15
M16
jiná permu tace Mi
jiná permu tace Mi
jiná permu tace Mi
m(i)
A
B
C
D
F
Mi
Ki
<<<s
A
B
C
D
H(i-1)
64 rund
H(i)
Schéma zpracování jednoho bloku zprávy kompresní funkcí
m(i)
H(i-1)
f
H(i)

Obr.: MD5

15. Kryptografické využití hašovacích funkcí

15.1. Standardy a protokoly symetrické a asymetrické kryptografie

Hašovací funkce se využívají jak v symetrické, tak asymetrické kryptografii a jsou součástí moderních standardů a protokolů, například v SSL/TLS, PKCS, v autentizačních schématech, zabezpečení integrity dat, v digitálních podpisech apod. Jejich odnoží jsou tzv. klíčované hašové autentizační kódy HMAC (Keyed-Hash Message Authentication Code), viz dále.

15.2. 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 haše postačí jedna podepisovací operace. Výjimkou jsou tzv. *kryptosystémy digitálního podpisu s obnovou zprávy*, kdy zpráva, která se podepisuje, bývá většinou velmi krátká, a systém ji podepisuje přímo.

15.3. 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.

15.4. Porovnání rozsáhlých databází

Uveďme si příklad banky, která ukládá všechna data ze všech účtů klientů do databázového systému, který je on-line zálohován, takže se vyskytuje současně na třech geograficky vzdálených místech. V určitý okamžik je nutné zjistit, zda jsou tyto databáze opravdu totožné. Mohli bychom například ze všech tří databází číst jednotlivé záznamy a porovnávat je proti sobě. To by znamenalo přenos celých databází komunikačním systémem, což by nemuselo být prakticky vůbec realizovatelné. Místo toho stačí na všech třech místech vypočítat otisky databází nebo jejich částí přenést hodnoty jen několika set bitů do centra. Pokud jsou hodnoty otisků shodné, máme jistotu, že se databáze neliší ani o jeden bit.

15.5. 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. Pokud útočník přečte haš uloženého hesla nějakého uživatele, není z této hodnoty díky jednocestnosti schopen určit uživatelovo heslo. Aby se vyloučil slovníkový útok, kdy si útočník předvypočítá haše pro často používaná slova a výrazy, používá se tzv. metoda solení. Při ní se vždy při ukládání otisku hesla vygeneruje i náhodný řetězec (sůl), který se poté hašuje dohromady s vlastním heslem. Do databáze se ukládá dvojice (sůl, hash(heslo, sůl)). Útočník by si pak musel předvypočítávat slovník teoreticky pro libovolnou hodnotu soli, což je výpočetně nemožné.

15.6. Pseudonáhodné funkce

Standard PKCS#5 umožňuje využít hašovací funkci k tvorbě "náhodného" šifrovacího klíče z passwordu pomocí pseudonáhodné funkce PRF jako klíč = PRF(password). Předpis je vidět z obrázku a vynecháme-li hodnotu soli, spočívá v hašování passwordu a následném mnohonásobném hašování výsledku. Počet hašování je dán konstantou c, jejíž hodnota se doporučuje minimálně 1000, ale používá se i 2000.
Výsledkem je krátký "náhodně vyhlížející" klíč DK, který je možné využít lépe než původní password. Jednak má pevnou délku a jednak z něho lze využít tolik bitů, kolik potřebujeme. Hodnota DK má pochopitelně lepší statistické vlastnosti než původní password. Tento postup se využívá ke tvorbě krátkých klíčů.

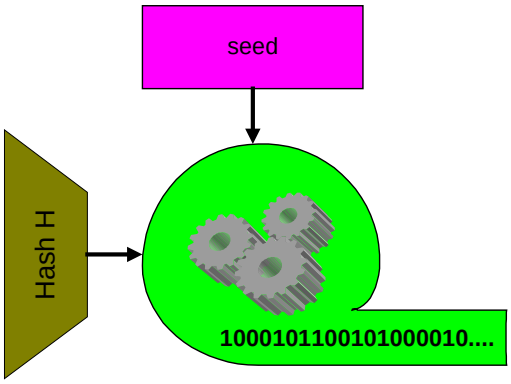
$T_1 = Hash\ (P \parallel S)$
$T_2 = Hash\ (T_1)$
…
$T_c = Hash\ (T_{c-1})$

šifrovací klíč $DK = T_c\langle 0..dkLen\text{-}1\rangle$

Obr.: Tvorba klíče DK z passwordu P podle PKCS#5 (S je sůl)

15.7. Pseudonáhodné generátory (PRNG)

Typické použití hašovacích funkcí jako pseudonáhodných generátorů je v případech, kdy máme k dispozici krátký řetězec (náhodných) dat (seed) s dostatečnou entropií. Může se jednat například o "krátký" 256bitový náhodný šifrovací klíč, záznam náhodného pohybu myši na displeji, časový profil náhodných stisků kláves apod. Přitom potřebujeme z tohoto vzorku získat pseudonáhodnou posloupnost o velké délce, například 1 GByte apod. A k promítnutí entropie původního vzorku (seedu) do delší posloupnosti se právě používají hašovací funkce.



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 - **většinou náhodným - nastavením** seed takto:
H(seed || 0x00000000), H(seed || 0x00000001), H(seed || 0x00000002), H(seed || 0x00000003), ...

Protože seed je náhodný, konstrukce bude pravděpodobně bezpečná a současnými útoky nedotčená.

15.8. Další příklady použití

Dalšími příklady mohou být kontrola šifrovacích klíčů při dešifrování (haš od hodnoty správného klíče může být uvedena například v hlavičce zašifrovaného souboru), autentizace (prokázání znalosti tajného klíče nepřímo pomocí haše), prokazování autorství (zveřejní se otisk dokumentu, dokument až ve vhodný čas později) apod.

16. Nejpoužívanější hašovací funkce

Dříve nejpoužívanějšími hašovacími funkcemi byly **MD2, MD4, MD5** [31], které mají 128bitový hašový kód. Od jejich používání se z bezpečnostních důvodů upouští. Důvodem není jen to, že 128bitový hašový kód je náchylnější na nalezení kolize (složitost 2^{64}), ale zejména to, že u MD2 a MD5 byla nalezena kolize kompresní funkce, tj.

f(H(i-1), **M(i)**) = f(H(i-1), **M(i)´**) a

u MD4 byla nalezena kolize už v roce 1996 H. Dobbertinem [33], viz obrázek.

****
CONTRACT
At the price of **$176,495** Alf Blowfish
sells his house to Ann Bonidea. .....

****
CONTRACT
At the price of **$276,495** Alf Blowfish
sells his house to Ann Bonidea. .....

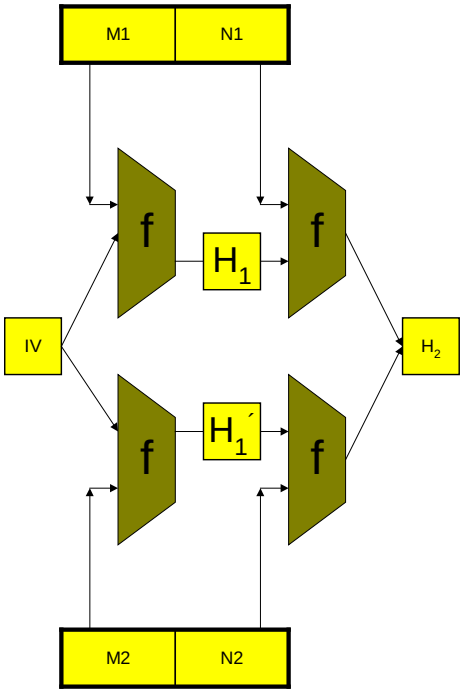
Zprávy, které mají stejný hašový kód MD4

Obr.: Kolize u MD4 [33]

V současné době jsou nejpoužívanějšími hašovacími funkcemi **MD5 a SHA-1** (se 160bitovým hašovým kódem). U obou byly však nalezeny vážné slabiny. U MD5 je možné generovat kolize i na osobním počítači v řádu hodin a u SHA-1 byla složitost nalezení kolize snížena na 2^{69} operací hašování, viz další odstavec.
V současné době se oblast hašovacích funkcí nalézá na křižovatce a nejlepší alternativou je použití hašovacích funkcí SHA-2, viz dále.

17. Průlom v kryptoanalýze hašovacích funkcí, zejména MD5 a SHA-1

V srpnu 2004 byly na rump session konference Crypto 2004 prezentovány kolize hašovacích funkcí MD4, MD5, HAVAL-128 a RIPEMD [34]. Dále se budeme věnovat pouze nejsložitější z uvedených funkcí, tj. MD5. Na konferenci nebyla zveřejněna metoda, ale pouze data a výsledky. Čínský tým [34] navrhl metodu nalézání **kolize dvou různých 1024bitových zpráv**. Praktické experimenty ukázaly dosažitelnost maximálně v řádu hodin na superpočítači. Metoda spočívá v nalezení dvou 1024bitových zpráv (M1, N1) a (M2, N2), které se liší o definovanou konstantu a mají stejnou haš, viz obrázek.



Obr.: Princip čínského útoku na MD5

Metoda tvorby kolizí byla ideově popsána až v březnu 2005 [35], ale bez podrobností, umožňujících reprodukovat útok. Nezávisle na tom byla na počátku března 2005 [36] vyvinuta a popsána [37] podobná metoda generování kolizí, rychlejší než [35] a umožňující generovat kolize i na notebooku. U SHA-1 byla v březnu 2005 oznámena metoda [40], umožňující nalézt kolizi SHA-1 se složitostí 2^{69} operací hašování.

Vývoj v této oblasti je velmi rychlý. V nejbližší době lze očekávat další zrychlení nalézání kolizí MD5 a SHA-1 i předložení kolize SHA-1 v historicky krátké době.

18. Hašovací funkce SHA-256, 384, 512 a 224

Z důvodu zvýšení odolnosti vůči kolizím je od 1. února 2003 k dispozici nová trojice hašovacích funkcí SHA-256, SHA-384 a SHA-512 [29] a od února 2004 SHA-224 (dodatek [29]). Tyto funkce přichází se zvýšením délky hašového kódu na 256, 384 a 512 bitů (SHA-224 má 224 bitový hašový kód), což odpovídá složitosti 2^{128}, 2^{192} a 2^{256} pro nalezení kolizí narozeninovým paradoxem. To je jednak už dost vysoká složitost a také to odpovídá složitosti útoku hrubou silou na tři délky klíčů, které nabízí standard AES.
Pokud se týká konstrukce nových funkcí, jsou velmi podobné SHA-1 a používají stejné principy, pracují však se složitějšími funkcemi a širšími vstupy. Podrobnosti lze nalézt v uvedených standardech. Jejich cílem bylo poskytnout větší odolnost proti kolizi a nabídnout odpovídající bezpečnost jako klíče pro AES.

19. Klíčovaný hašový autentizační kód - HMAC

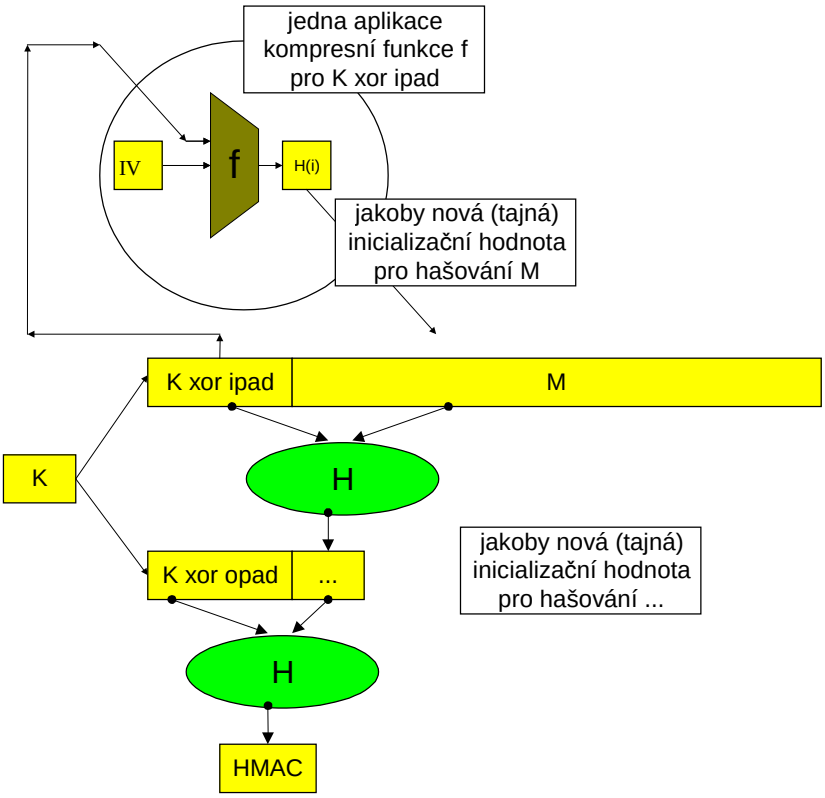
Klíčované hašové autentizační kódy zpráv HMAC zpracovávají hašováním nejen zprávu M, ale spolu s ní i nějaký tajný klíč K. Jsou proto podobné autentizačnímu kódu zprávy MAC, ale místo blokové šifry využívají hašovací funkci. Používají se jak k nepadělatelnému zabezpečení zpráv, tak k autentizaci (prokazováním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

19.1. Obecná definice HMAC

HMAC je definován ve standardu FIPS PUB 198 [32], kde je o něco obecněji popsán, než v RFC 2104 a ANSI X9.71. Jeho definice závisí na tom, kolik bajtů (B) má blok kompresní funkce. Například u SHA-1 je B = 64, u SHA- 384 a SHA-512 je B = 128.
Definujeme konstantní řetězce *ipad* jako řetězec B bajtů s hodnotou 0x36 a *opad* jako řetězec B bajtů s hodnotou 0x5C. Klíč K doplníme bajty 0x00 do délky B a definujeme hodnotu HMAC jako

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

kde || označuje zřetězení. Kolize HMAC nejsou kolizemi použité funkce ohroženy, neboť dnes se neumí nalézt kolize pro utajenou inicializační hodnotu. 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)

Klíčovaný hašový autentizační kód zprávy HMAC-H(K, M) je funkčně podobný autentizačnímu kódu zprávy MAC, ale místo blokové šifry využívá hašovací funkci (H). Označuje se konkrétně podle toho, jakou hašovací funkci používá, např. HMAC-SHA-1(K, M). M označuje zprávu a K klíč. Je definován ve standardu FIPS 198 (kde je popsán o něco obecněji než v RFC 2104 a ANSI X9.71) a jeho definice závisí na délce bloku kompresní funkce v bajtech (např. u MD5/SHA-1/SHA-256 je to B = 64 bajtů, u SHA-384/SHA-512 je to B = 128 bajtů) a na délce hašového kódu hašovací funkce H. HMAC používá dvě konstanty, a to *ipad* jako řetězec B bajtů s hodnotou 0x36 a *opad* jako řetězec B bajtů s hodnotou 0x5C. Klíč K se doplní nulovými bajty do plného bloku délky B a poté definujeme HMAC-H(K, M) = H((K xor *opad*) || H((K xor *ipad*)|| M)), kde || označuje zřetězení. Na obrázku je schéma HMAC.
Tajný klíč se modifikuje konstantou *ipad* a výsledek (K xor *ipad*) tvoří začátek vstupu do hašování. Je to definováno tak, že K xor *ipad* je přesně jeden blok kompresní funkce, takže po jeho zpracování dostáváme kontext H_1. Následuje zpracování zprávy M, čili její hašování jakoby začínalo z (útočníkovi neznámé) inicializační hodnoty IV = H_1, bez uvažování předsazeného řetězce (K xor *ipad*). Tento princip se použije ještě jednou, ale nikoli na zprávu jako takovou, nýbrž na obdrženou haš. Uvědomme si, že stačí nalézt jen kolizi pro H((K xor *ipad*)|| M), protože ta se automaticky projeví v celém HMAC.

Po publikování [35] víme, že se tato práce nijak netýká kolize pro tajné nastavení inicializační konstanty, proto **konstrukci HMAC považujeme za nedotčenou současnými útoky**. Avšak odhaduje se, že i při tajné inicializační hodnotě by nalezení kolize mohlo být výpočetně méně náročné než by mělo teoreticky mělo být.
Proti použití prolomených hašovacích funkcí ve funkci HMAC v současné době není námitek, neboť není známo žádné oslabení funkce autentizačního kódu. Je však nutné sledovat, zda se útok neprohloubí i na tajné inicializační hodnoty.

19.2. Nepadělatelný integritní kód, autentizace původu dat

Zabezpečovací kód HMAC-SHA-1(M, K), pokud je připojen 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.

19.3. Průkaz znalosti při autentizaci entit

HMAC může být použit 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.

20. Generické problémy (současných) iterativních hašovacích funkcí

Generické problémy současných hašovacích funkcí ukazují dvě práce. První představil Joux [38] na konferenci Crypto v srpnu 2004 a druhý Kelsey-Schneier [39] v listopadu 2004. **Obě dvě práce ukazují, že iterativní konstrukce hašovací funkce implikuje značnou odlišnost této funkce od náhodného orákula.** To má značný dopad na praxi, neboť předpokládaně bezpečné hašovací funkce třídy SHA-2 jsou iterativní a mají proto tuto slabinu.

Joux ukazuje, že

1) u iterativních hašovacích funkcí lze docílit **mnohonásobné kolize** mnohem jednodušeji než ve srovnání s náhodným orákulem
2) **kaskádovitá konstrukce** F || G pomocí dvou hašovacích funkcí pozbývá smyslu, neboť očekávaná složitost nalezení kolize není součinem dílčích složitostí, ale spíše součtem

Kelsey-Schneierova práce

1) obsahuje výrazně zlepšenou metodu konstrukce multikolizí oproti Jouxově práci,
2) umožňuje konstruovat druhý vzor zprávy u iterativních hašovacích funkcí se složitostí cca $2*2^{n/2} + 2^{n-k+1}$ pro velmi dlouhé zprávy o délce 2^k blízké $2^{n/2}$.

Konkrétně pro SHA1 lze ke zprávě o délce 2^{60} bajtů vytvořit druhý vzor se složitostí 2^{106} na rozdíl od teoretické složitosti 2^{160}.

Vzhledem k uvedeným vlastnostem je pravděpodobné, že se bude hledat nový koncept hašovacích funcí.
O generických problémech hašovacích funkcí je blíže pojednáno v [30].

21. Literatura

[25] Special Publication SP800-38A: *Recommendation for Block Cipher Modes of Operation - Methods and Techniques*, NIST, http://csrc.nist.gov/

[26] W. Diffie, M. Hellman, *Privacy and Authentication: An Introduction to cryptography*, Proceedings of the IEEE, 67 (1979), pp. 397 - 427

[27] *PKCS#5 v2.0: Password-Based Cryptography Standard*, RSA Labs, March 25, 1999

[28] Special Publication SP800-38C: *Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality*, http://csrc.nist.gov/, May, 2004

[29] *FIPS PUB 180-2: Secure Hash Standard*, NIST, http://csrc.nist.gov/

[30] V. Klíma: Hašovací funkce, principy, příklady a kolize, přednáška na semináři Cryptofest, 19.3. 2005, http://cryptography.hyperlink.cz/2005/cryptofest_2005.htm.

[31] *MD2, MD4, MD5: RFC 1319, 1320, 1321*, http://www.rfc-editor.org/

[32] *FIPS PUB 198: HMAC*, NIST, http://csrc.nist.gov/CryptoToolkit/tkhash.html, viz též RFC 2104, http://www.rfc-editor.org/

[33] H. Dobbertin, *Cryptanalysis of MD4*, FSE 1996, pp. 53 - 69

[34] X. Wang, D. Feng, X. Lai, H. Yu, "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD", rump session, CRYPTO 2004, *Cryptology ePrint Archive*, Report 2004/199, http://eprint.iacr.org/2004/199

[35] Xiaoyun Wang and Hongbo Yu: How to Break MD5 and Other Hash Functions, March 6, 2005, http://www.infosec.sdu.edu.cn/paper/md5-attack.pdf.

[36] V. Klima: Finding MD5 Collisions – a Toy For a Notebook, *Cryptology ePrint Archive*, Report 2005/075, http://eprint.iacr.org/2005/075.pdf, March 5, 2005

[37] V. Klima: Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications, March 31, 2005, IACR ePrint archive, Report 2005/102, http://eprint.iacr.org/2005/102.pdf

[38] A. Joux: Multicollisions in iterated hash functions. Application to cascaded constructions. Proceedings of Crypto 2004, LNCS 3152, pages 306-316.

[39] John Kelsey, Bruce Schneier: Second Preimages on n-bit Hash Functions for Much Less than 2^n Work, http://eprint.iacr.org/2004/304/, November 15, 2004

[40] Wang X., Yin L., Yu H.: Collision Search Attacks on SHA1, February 13, 2005, http://theory.lcs.mit.edu/~yiqun/shanote.pdf