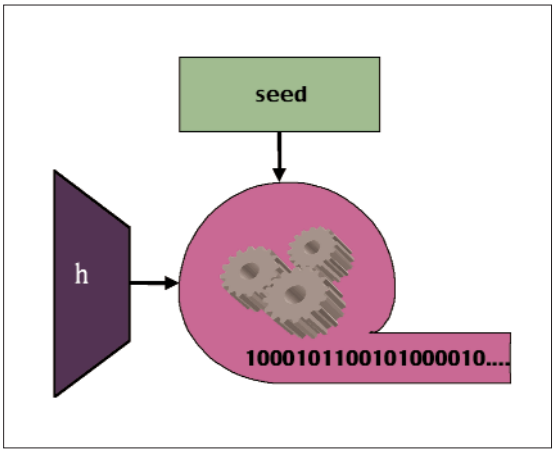
Kryptologie pro praxi – hašovací funkce jako PRNG

Hašovací funkce jsou silným nástrojem moderní kryptologie. Jsou jednou z klíčových kryptologických myšlenek počítačové revoluce a přinesly řadu nových použití. Kromě tradičních využití jednocestnosti a bezkoliznosti si vývojáři také povšimli vlastností náhodnosti a nepredikovatelnosti jejich výstupů. V tomto článku ukážeme jak hašovací funkce v praxi použít pro generování pseudonáhodných posloupností vysoké kvality.

Bezpečnost

Z hlediska bezpečnosti bychom byli rádi, kdyby se hašovací funkce chovala jako náhodné orákulum, tj. stroj, který na nový vstup odpovídá náhodným výběrem z množiny možných výstupů (jeden výstup se tedy může i opakovat), přičemž na tentýž (starý, opakovaný) vstup odpovídá tímtéž výstupem. I když z teoretického hlediska nejsou hašovací funkce náhodná orákula, prakticky se tak jeví. Každá změna, byť i jednoho bitu na vstupu, má za následek nepredikovatelnou náhodnou změnu všech bitů na výstupu s pravděpodobností 1/2. A naopak jakákoliv změna na výstupu by měla vést k nepredikovatelné a náhodné změně na vstupu (vstupech). Náhodnosti a nepredikovatelnosti se začalo využívat ke konstrukci pseudonáhodné funkce (*PRF*), a tu pro generátory pseudonáhodných čísel (*PRNG*). Poznamenejme, že oslabení vlastnosti bezkoliznosti hašovacích funkcí (kolizi MD5 lze nalézt za hodinu, kolizi SHA-1 se složitostí 2^{66} operací) oslabilo i naši důvěru v jejich použití pro *PRF*/*PRNG* (i *HMAC*), i když tento fakt, jak bylo řečeno v článku ve *ST 12/2004*, nemá přímý vliv na oslabení vlastností funkcí *PRF* a *HMAC* (o *HMAC* viz *ST 02/2004*).



Obr. 2 Hašovací funkce v konstrukci PRNG

Používejte SHA-256 nebo HMAC-SHA-256

Hašovací funkce h zpracovává prakticky neomezeně dlouhá vstupní data M na krátký výstupní hašový kód $h(M)$. Starší normy pro *PRF*/*PRNG* byly psány pro MD5/SHA-1, budeme místo nich ale raději uvažovat SHA-256 nebo SHA-512 [1]. Uvidíme také, že místo funkce h se v generátorech také používá *HMAC-h*, což je samozřejmě robustnější, ale i bezpečnější (je to ještě větší obrana proti využití kolizí). Připomeňme, že *HMAC-h*(M, K) = h ((K xor *opad*) || h((K xor *ipad*)|| M)), kde || je zřetězení, K klíč, M zpráva a h příslušná hašovací funkce.

$$T_1 = h(P \,||\, S)$$
$$T_2 = h(T_1)$$
$$\ldots$$
$$T_c = h(T_{c-1})$$

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

Obr. 1 Tvorba klíče DK z passwordu P podle PKCS#5

Jak tvořit náhodné klíče

Často potřebujeme odvodit 128 nebo 256bitový klíč z passwordu nebo klíčové fráze, které mohou mít různou délku a různou entropii. Generování takto krátkých klíčů z passwordu (délku klíče v bajtech označme *dklen*) řeší například standard PKCS#5 [2] pomocí pseudonáhodné funkce PBKDF1. Její předpis je vidět z obrázku a spočívá v hašování passwordu (P) se solí (S) 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“ řetězec DK = PBKDF1 (P, S, c, *dklen*), z něhož můžeme použít tolik bitů, kolik potřebujeme, a má pochopitelně lepší statistické vlastnosti než původní password.

Může hašovací funkce degenerovat?

V předchozí konstrukci se používá výraz $h^{1000}(x)$ a my tušíme, že každé hašování nám odebírá trochu entropie z původního řetězce x. Nechť hašovací funkce h má $n$bitový výstup a označme $p(t)$ část $n$bitových hodnot, které se mohou objevit jako výstup funkce $h^t(x)$. Uvádí se přibližný odhad $p(1) = 1 - 1/e = 0{,}632$, $p(2) = 0{,}469$, $p(3) = 0{,}374$, ..., $p(t+1) \sim 1 - e^{-p(t)}$. Pro $t = 2^k$ ($1 << k << n/2$) pak přicházíme zhruba o k bitů entropie. Při hašování 1000krát přicházíme asi o 10 bitů entropie. Je nutné si uvědomit, že toto je heuristický a teoretický odhad, a že případný útočník ho bude ztěží využívat, pokud původní entropie bude kolem řekněme stovky bitů. Ke ztrátě entropie sice dojde, ale útočník nebude moci zjistit jakým způsobem a tento fakt využít. Pokud bude entropie vzorů malá, řekněme kolem 10–20 bitů, útočník může všechny vzory vyzkoušet, pak experimentálně zjistit jak množina obrazů $h^t(x)$ degeneruje skutečně a poté toho faktu eventuálně využít.

Pseudonáhodné generátory na bázi haší

Typické použití hašovacích funkcí jako pseudonáhodných funkcí pro *PRNG* nastává v případech, kdy máme k dispozici krátký řetězec 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 GB apod. Tak lze například z 256bitového klíče vygenerovat dlouhou heslovou posloupnost pro proudovou šifru. A k promítnutí entropie původního vzorku (*seedu*) do delší posloupnosti se právě používají hašovací funkce. Hlavní ideový rozdíl oproti minulému použití je v tom, že vstupem je náhodný zdroj a výstupem obvykle dlouhá posloupnost.

PRNG pro náhodný seed

Například standard PKCS#1 v.2.1 [5] definuje pseudonáhodný generátor MGF1 (Mask Generation Function) pomocí hašovací funkce h s počátečním – většinou náhodným – nastavením *seed* jako h(*seed* || 0x00000000), h(*seed* || 0x00000001), h(*seed* || 0x00000002), h(*seed* || 0x00000003), ..., kde 0x... znamená hexadecimální vyjádření daného čísla jako čtyřbajtové hodnoty. Protože *seed* je náhodný, nedají se současné útoky na hašovací funkce zde využít. Tento předpis je velmi jednoduchý a přitom bezpečný. Jednomu návrháři se zdál příliš jednoduchý, a tak pro generování další hodnoty (i+1) využil místo *seedu* ve funkci h složitější hodnotu, a to právě vypočítanou $h(i)$= = h(*seed* || INT(i)). Jak snadno nahlédnete, $h(i+1) = h(h(i)$ || INT(i)) a u této posloupnosti stačí znát jednu generovanou hodnotu, aby se určily všechny následující, což není dobré. Na generickou odlišnost velké většiny hašovacích funkcí od náhodného orákula se někdy zapomíná,

neboť z žádného výstupu náhodného orákula by neměl jít vypočítat jakýkoliv jiný výstup, zatímco pro velkou většinu hašovacích funkcí lze z $h(M)$ vypočítat $h(M^*)$, kde $M^* = M \,||\, definitorický\ doplněk \,||\, N$, kde N může být libovolný řetězec. Při hašování zprávy M^* můžeme totiž vyjít z toho, že po průběžném zhašování $M \,||\, definitorický\ doplněk$ dostáváme hodnotu $h(M)$, kterou známe, a tudíž vlastně od ní (namísto od inicializačního vektoru) začínáme hašovat řetězec N a obdržíme h(M*), i když počátek zprávy M^* (tj. M) vůbec nemusíme hašovat a dokonce ani znát. Na tuto vlastnost dejte proto při vlastních konstrukcích velký pozor.

PRNG pro nenáhodný seed

Tam, kde *seed* není zcela náhodný (je to například password), se používá komplikovanější postup, viz například funkce *PBKDF*2(*P*, *S*, *c*, *dklen*) z PKCS#5. Ta na základě passwordu P a soli S generuje pseudonáhodnou posloupnost $T1$, $T2$, $T3$, ..., kde Ti, je vždy součet (xor) sloupce v následující tabulce. Počet řádků v tabulce odpovídá počtu iterací (c), je to konstanta, jejíž doporučené hodnoty jsou 1000 nebo 2000.

Tab. 1 Pseudonáhodný generátor PBKDF2(P, S, c, dklen) podle PKCS#5

$U_1 = h(P, S, 0x00000001)$	$U_1 = h(P, S, 0x00000002)$	$U_1 = h(P, S, i\ (4\ B))$
$U_2 = h(P, U_1)$	$U_2 = h(P, U_1)$	$U_2 = h(P, U_1)$
$U_3 = h(P, U_2)$	$U_3 = h(P, U_2)$	$U_3 = h(P, U_2)$
....	....	....
$U_c = h(P, U_{c-1})$	$U_c = h(P, U_{c-1})$	$U_c = h(P, U_{c-1})$
T_1 = xor *sloupce*	T_2 = xor *sloupce*	T_i = xor *sloupce*

Existuje více norem pro *PRF*/*PRNG*. Uvedeme příklady. Například standard pro podpisové schéma DSA [4] definuje náhodný generátor, jehož definici bychom mohli zestručnit na tvar $x_0 = K$, kde K je počáteční vstup (klíč), a $x_j{+}1 = h((K + 1 + x_j) \bmod 2^b)$, kde b je délka bloku hašovací funkce h. Další velmi podobný postup definuje standard PKCS#12 [3]. Tyto konstrukce jsou robustní a jsou považovány za bezpečné.

Závěr

Seznámili jsme se s několika možnostmi jak využít hašovací funkci jako pseudonáhodnou funkci (*PRF*) pro potřeby generování pseudonáhodných posloupností. Tyto techniky nejsou dotčeny nedávnými objevy kolizí u řady hašovacích funkcí. Do budoucna by však bylo prozíravější používat i v těchto technikách dosud neprolomené hašovací funkce.

Vlastimil Klíma, Tomáš Rosa,
v.klima@volny.cz, trosa@ebanka.cz

LITERATURA

[1] FIPS PUB 180-2, Secure Hash Standard (SHS), August 2002 (SHA-1, SHA-256, SHA-384, and SHA-512), http://csrc.nist.gov/CryptoToolkit/tkhash.html

[2] PKCS #5: Password-Based Cryptography Standard, ver. 2.0., RSA Labs, http://www.rsasecurity.com/rsalabs/node.asp?id=2127

[3] PKCS #12: Personal Information Exchange Syntax Standard, ver. 1.0, RSA Labs, http://www.rsasecurity.com/rsalabs/node.asp?id=2138

[4] FIPS PUB 186-2, Digital Signature Standard (DSS), NIST, 5.10.2001 (poslední změna), http://csrc.nist.gov/cryptval/

[5] PKCS #1: RSA Cryptography Standard, ver. 2.1., RSA Labs, http://www.rsasecurity.com/rsalabs/node.asp?id=2125

[6] E-archivy http://cryptography.hyperlink.cz, http://crypto.hyperlink.cz

Paradox informační společnosti

Koncem letošního května v kongresovém sále Masarykovy koleje v Praze proběhl, za účasti a zájmu předních odborníků z praxe i ze specializovaných akademických pracovišť, první ročník konference o řízení informačních a komunikačních technologií – ICTM 2005. Představa politických a mnohdy i ekonomických manažerů o fungující vyspělé informační společnosti v řadě případů celkem legračně předpokládá, že našim národním i evropským cílem je, aby si každá babička dokázala prostřednictvím portálu státní správy spočítat výši penze, a aby také uměla komunikovat s vlastními vnuky e-mailem. Organizují se v tomto duchu nejrůznější kurzy s finanční podporou státu.

V řadě soukromých podniků (o těch se státní účastí ani nemluvě) se problémy s řízením ekonomických a výrobních procesů řeší nákupem nových výpočetních kapacit a návratnost vynaložených prostředků se kamufluje všelijak. Mnohdy slýcháme z úst vrcholového vedení podniků, že investice do informačních a komunikačních technologií jsou nezbytné především pro udržení konkurenční výhody a udržení pozice podniku na globálním trhu. Specialisté podnikových oddělení IT jsou proto obvykle slušně honorováni, ale také v průběhu normální pracovní doby přetěžováni řešením víceméně banálních problémů firemní administrativy s programovým vybavením referentů. Při náporech požadavků na uživatelskou podporu, při aktivitách spojených s technologickou profylaxí podnikové infrastruktury ICT a při zapojení všech existujících intelektuálních kapacit do vertebrálních projektů rozvoje podniku, se nelze ani příliš divit. Zavedení pořádku, jednotných pravidel i pokusy o standardizaci řízení vnitropodnikových procesů, jsou v takové situaci úkolem značně komplikovaným a nesystematicky se mnohé koncepce diskutují až po pracovní době obvykle nedůsledně a někdy i nekompetentně.

Jedním z cílů konference ICTM 2005 bylo poskytnout účastníkům komplexní přehled o existujících standardních metodách řízení informačních a komunikačních technologií, jako jsou např. ITIL, CobiT, SixSigma a další, které diskutované problémy ve výrobní a podnikatelské sféře i ve státní správě dokáží v obecné rovině s úspěchem řešit. Byly představeny ukázky praktických implementací a výsledků, které se postupně promítly i do ekonomiky konkrétních podniků. Příčiny i důsledky těchto jevů diskutovali ve dvou panelových diskuzích přední specialisté a odborníci zabývající se aktivně implementacemi jednotlivých kocepcí. Jednání konference provázel sborník s úplným zněním všech příspěvků.

Generálním partnerem setkání byla společnost Computer Associates, partnery Česká společnost pro jakost, HP, LBMS, CRUX IT, Omnicom a další. Mediálními partnery byly časopisy CLICK, Convergence, Inside, IT Systems, Professional Computing a Sdělovací technika. Podrobnosti lze nalézt např. na www.itil.cz nebo na webové stránce katedry telekomunikační techniky ČVUT-FEL.

Jik