Jak ze čtverce udělat kruh

V tomto článku vás seznámíme s novým využitím blokové šifry jak pro generování náhodných čísel, tak pro šifrování. Ukážeme vám, jak lze blokovou šifru výhodně přeměnit na proudovou pomocí běžného čítače. Tento nový operační modus byl oficiálně navržen americkým standardizačním úřadem NIST.

Kvalitní pseudonáhodný generátor zadarmo

Pokud jste někdy potřebovali rychle nějakým způsobem generovat náhodná data pro svůj program, určitě jste si byli vědomi toho, že jakákoliv nevhodná odchylka generovaných dat může cíl tohoto programu zhatit. Často je prostě potřeba mít jistotu, že data jsou pořádně "zamíchaná" a nemají žádné strukturální vlastnosti (pokud vám postačí lineární kongruentní generátory, podívejte se např. na [1]). Často je ale na dosah ruky nějaká knihovna s proudovými nebo blokovými šiframi a stačí ji jen použít. Zdrojové kódy šifer jsou totiž volně dostupné na internetu, například na [2], takže zbývá jen vědět, jak na to. Ukážeme si, že je to vlastně velmi jednoduché.

Proudové šifry jako generátory náhodných znaků

Proudové šifry (jako všechny ostatní šifry) používají nějaké tajné nastavení - šifrovací klíč, na němž závisí celá jejich výstupní posloupnost. Ta se nazývá heslo, které se obvykle xoruje bajt po bajtu na otevřený text. Pokud by někdo v tomto hesle objevil nějaké strukturální závislosti včetně sebemenších statistických odchylek, narazil by ve skutečnosti na slabinu šifry, což by vedlo k jejímu odsunu na pohřebiště neúspěšných, rozbitých šifer. U kvalitních šifer předpokládáme, že jejich výstupní posloupnost je posloupností nezávislých náhodných znaků. Stačí proto zvolit nějaký ani ne tajný klíč a z dané šifry tuto posloupnost používat v roli generátoru náhodných znaků. Velmi rychlá je například šifra RC4 (viz [3]), jejíž zdrojový kód můžete vidět na obrázku 1. Z posledních prací vyplývá, že jejích prvních 512 (pro puristy 3072) bajtů hesla h(i) je vhodné "zahodit" a používat až další produkci, neboť určité statistické nuance zde nalezeny byly. Dále byly nalezeny určité strukturální závislosti i ve velmi dlouhém výstupu, ale jedná se o velmi složité vztahy mezi nejnižšími bity výstupních bajtů, přičemž desítky megabajtů zde hrají roli základních měrných jednotek pro měřitelné odchylky. Běžné aplikaci to pochopitelně nevadí, ale v případě, že chceme "ještě kvalitnější" výstup, můžeme použít některou kvalitní blokovou šifru.

Blokové šifry jako generátory náhodných znaků

Blokové šifry lze ke generování náhodných znaků použít podobným způsobem jako proudové, jde jen o to, čím plnit jejich vstup a jak využívat výstup, čili o tzv. operační modus. Zvolíme si nějaký klíč pro vybranou šifru, řekněme AES (viz [4]), a nyní ji budeme chtít využít tak, aby nám poskytovala proud znaků. Máme na vybranou "starý" modus OFB (přehledově o modech viz [7]), nebo "nový" čítačový modus. Jsou skoro stejně staré, ale o čítačovém modu se dlouho nehovořilo, i když ho Diffie s Hellmanem publikovali už v roce 1979 (viz [5]). Připomeňme si tedy Output Feedback (OFB) neboli modus zpětné vazby z výstupu šifry. Pokud naši šifru AES se zvoleným klíčem K označíme jako funkci f, jde o to naplnit její vstup na počátku nějakou inicializační hodnotou (IV, je to 128bitový řetězec) a tuto hodnotu "točit přes šifru stále dokola". Formulkou by to bylo takto: x(0) = IV, x(1) = f(x(0)), x(2) = f(x(1)),... , tedy zkráceně x(i+1) = f(x(i)), přičemž hodnoty x(i), i= 1, 2, ... bychom použili jako náhodná čísla.

Abychom se nezamotali příliš brzo

Je zřejmé, že v obdržené posloupnosti nemůžeme pokračovat donekonečna, protože možných binárních řetězců délky 128 bitů je jen 2128. Zákonitě se tedy v generované posloupnosti x(i) musíme jednou dostat do bodu, kde jsme už jednou byli. Protože funkce f je reverzibilní ("zpětný chod" je vlastně dešifrování), je zřejmé, že do každého bodu x(i) vede jen jeden předchůdce, neboli vzniká čistý cyklus bez "přívodních větví", viz obr. 2. Protože f je náhodné zobrazení, celá množina možných řetězců x(i) se rozpadá do řady takových náhodných cyklů. Z teorie pravděpodobnosti pak vyplývá, že průměrná délka cyklu je 2127. To je určitě pro mnoho aplikací, kde jsou potřeba náhodná data, dostatečné. Pro šifrování chceme mít ale jistotu o něco větší. Ostatně nešifrujeme data jen tak pro nic za nic, takže bychom přivítali, kdybychom o délce tohoto cyklu mohli něco tvrdit. Může se totiž stát, že se náhodně trefíme do cyklu o délce 2 nebo tisíc bloků, což je velmi málo, a došlo by k tzv. dvojímu použití hesla, o němž

budeme hovořit dále. Následující trik ale převádí všechny body z obrázku 2 do jediného (!) cyklu o délce přesně 2128.

Čítač v šifrování

Běžný čítač je velmi jednoduchá funkce, kdy se obsah daného registru, v našem případě 128bitového, zvýší o jedničku, přičemž ze stavu (hex.) 0xFF...FF se přechází do nulového stavu. Triviálně vidíme, že tento čítač má periodu 2128, ať začneme z jakéhokoliv počátečního bodu x(0) = IV. Takže čítač projde všechny různé 128bitové stavy. To bychom přesně potřebovali od šifry. Trik je v tom, že jako výstup nepoužijeme přímo stavy čítače, ale jejich šifrové obrazy. Dostáváme ale jediný cyklus? Protože funkce f je reverzibilní, nemůže se stát, že by obrazy dvou různých stavů čítače byly stejné, protože pak i jejich vzory, získané odšifrováním, by musely být stejné. Maximální délka posloupnosti tedy zůstává 2128 a navíc posloupnost šifrových obrazů se u kvalitní šifry jeví jako dostatečně kvalitní zdroj náhodných znaků. Je tu určitá teoretická výjimka, která nám asi v praxi nebude příliš vadit. Naše nová posloupnost je přece jen "pravidelná", a to sice v tom, že právě nikdy během generování 2128 128bitových bloků nedojde k tomu, že by byly vygenerovány dva stejné bloky. U náhodné posloupnosti to téměř jistě nastane, a to podle tzv. narozeninového paradoxu už s 50% pravděpodobností někdy v prvních 264 blocích. V jiném pohledu však u kvalitní šifry nejsme schopni tuto posloupnost odlišit od posloupnosti náhodných znaků. Kdybychom mezi těmito obrazy nalezli jakékoliv využitelné statistické nebo analytické vztahy, danou šifru bychom tím vlastně "rozbili" nebo odhalili její slabosti.

Dvojí použití hesla

Za okamžik si uvedeme oficiální a formální definici čítačového modu. Nyní si připomeneme, oč vlastně z kryptografického hlediska jde. Pokud používáme stejný klíč, viděli jsme, že dostáváme jedinou funkci f, a tím i jediný cyklus, v kterém se pohybujeme. Hodnoty f(i) používáme přímo jako heslo, které xorujeme na otevřený text. Pokud bychom použili pro dvě zprávy stejnou inicializační hodnotu pro čítač, obdrželi bychom i stejný proud hesla. Označíme-li proud bajtů hesla, otevřených a šifrových textů h(j), OT1(j), OT2(j) a ŠT1(j), ŠT2(j), pak bychom v tomto případě dostali ŠT1(j) = OT1(j) [197] h(j) a ŠT2(j) = OT2(j) [197] h(j). Z toho jednoduchou úpravou vyplývá, že ŠT1(j) [197] ŠT2(j) = OT1(j) [197] OT2(j). Vidíme, že heslo z tohoto vztahu zcela vypadlo, přičemž hodnotu šifrových textů známe. Známe proto i hodnotu proudu bajtů OT1(j) [197] OT2(j). Nyní použijeme například metodu předpokládaného slova, znázorněnou na obrázku 4. Je to ve skutečnosti prosté vyzkoušení hodnoty OT1(j) nebo OT2(j). Ze znalosti výrazu OT1(j) [197] OT2(j) pak dopočítáme vždy druhý otevřený text. Pokud vyjdou nesmysly, zkoušíme jiné slovo nebo jinou pozici tak dlouho, až nám vyjde smysluplný text. Získané texty v OT1 i v OT2 pak rozšiřujeme na obě strany, až dostaneme oba dva otevřené texty v plné délce. Je tedy zřejmé, že dvojímu použití hesla musíme zabránit i v případě čítačového modu. Zde vidíme velmi jasně, že je potřeba zajistit, aby hodnoty IV1 a IV2, které inicializují příslušné úseky hesla, byly dostatečně vzdálené, aby nedošlo k překryvu na obrázku 5.

Co na to norma

Norma velmi přísně stanoví, že pokud se k šifrování používá tentýž klíč K s různými inicializačními hodnotami pro různé zprávy, nikdy nesmí dojít k překryvu hesla za celou dobu platnosti klíče K. Norma umožňuje různé postupy inkrementace čítače, například může probíhat jen v dolní m-bitové části b-bitového čítače (tj. mod 2m). Dále je možné čítač (nebo jeho dolních m bitů) považovat za lineární posuvný registr se zpětnou vazbou a jeho stavy obnovovat posunem apod. Dále je tu zajímavá možnost nastavit horních b-m bitů čítače na hodnotu, která je nějak jednoznačně spojena se zprávou, jež se šifruje (napadá vás možná b-m bitů haše zprávy), zatímco zbylých m bitů se klasicky inkrementuje. Těmito všemi možnými opatřeními se docílí podmínky, aby nikdy nebylo použito totéž heslo. Úplný popis čítačového modu naleznete ve speciální publikaci NIST [6], která přejímá definice i dalších čtyř klasických modů (ECB, CBC, CFB a OFB) tak, jak byly definovány v osmdesátých letech v normě FIPS 81.

Jedna výhoda za všechny

Oproti výše jmenovaným čtyřem klasickým modům má čítačový modus jednu neobyčejnou výhodu. Můžeme pomocí něho nezávisle na čemkoli odšifrovat nebo zašifrovat jakýkoliv bajt na jakékoliv pozici daného souboru nebo proudu dat, a to velmi rychle. Postačí vypočítat odpovídající hodnotu čítače a z ní jedním voláním funkce f obdržet příslušné heslo. U modu OFB bychom museli počítat všechny stavy funkce f, abychom ji dostali do odpovídající pozice. U modu CBC a CFB bychom zase potřebovali znát okolí daného bajtu šifrového textu, abychom mohli správně nastavit zpětnou vazbu. To například u databázových aplikací nemusí být vždy možné, protože náš šifrovací/dešifrovací modul může dostat někdy jen část šifrovaného záznamu. Čítačový modus je v tomto ohledu velmi jednoduchý, jeho

nevýhodou je nutnost splnit podmínku různých čítačů. V praxi je to nejlépe možné udělat tak, že inicializační hodnotu generujeme náhodně.

Jedna nevýhoda za všechny

Vlastností jak blokových, tak proudových šifer je, že (alespoň v základním tvaru) nezajišťují integritu dat. Data sice utají, tj. poskytují důvěrnost, ale proti jejich modifikaci obecně nic moc nezmohou. Tato vlastnost bývá tak často opomíjena, že se téměř vžilo, že když je něco zašifrované, je to bezpečné. To je pochopitelně zásadní omyl. Pokud změníme šifrový text u blokové šifry (kromě modu OFB a čítačového modu, které právě přetváří blokovou šifru na proudovou), dojde ke špatnému odšifrování ve dvou blocích otevřeného textu. Naproti tomu u čistě proudové šifry dojde ke špatnému odšifrování pouze ve znaku, který odpovídá místu změny v šifrovém textu. Toho lze pochopitelně zneužívat k různým útokům, pokud útočník ví, jaký druh informace se kde nachází (například v souborech bankovních transakcí, databázích apod.). Vzhledem k tomu, že změna [197] D na šifrovém textu vede ke změně [197] D v otevřeném textu, útočník může snadno otevřený text měnit, aniž by znal příslušný šifrový klíč.

Shrnutí

Seznámili jsme se s novým operačním modem blokových šifer, čítačovým modem. Jeho výhodou je snadnost použití i jednoduchost výpočtu hesla pro danou pozici otevřeného/šifrového textu. Lze jej využít jak k proudovému šifrování, tak jako zdroj náhodných znaků pro modelování různých situací, které často vznikají při programování různých aplikací.

Vlastimil Klíma, autor@chip.cz

Literatura: [1] Klíma, V.: Generátory náhodných čísel I až IV, Chip 3 - 6/98 [2] zdrojové kódy šifer: ftp://ftp.funet.fi/pub/crypt/cryptography/symmetric/ [3] Klíma, V.: RC4: Šifra, která míchá karty, Chip 9/99, str. 42 - 44 [4] Klíma, V.: AES - Nová šifra nastupuje, Chip 5/02, str. 142 - 144 [5] Diffie, W., Hellman, M. E.: "Privacy and authentication: An introduction to cryptography", Proceedings of the IEEE, 67 (1979), 397 - 427 [6] Recommendation for Block Cipher Modes of Operation, Methods and Techniques, NIST Special Publication 800-38A, 2001, dostupné na http://csrc.nist.gov/encryption/tkmodes.html [7] Klíma, V.: Šifry s mnoha tvářemi, Chip 7/00, str. 50 - 53 [8] Elektronický archiv uvedených i dalších článků: http://www.decros.cz/bezpecnost/_kryptografie.html

Pár tipů pro programátory

Představme si, že pro nějakou hru (program) potřebujeme generovat náhodná čísla 0 až 9, přičemž by bylo vhodné nabídnout uživateli tutéž hru (například totéž rozdání karet) si zahrát ještě jednou. Za prvé to znamená umět vygenerovat náhodná čísla, za druhé umět to přesně zopakovat. Řešení vás už určitě napadá - prostě číslo hry bude rovno inicializačnímu vektoru a klíč bude někde v programu nastaven na konstantu, nebo obráceně - klíč bude číslo hry a IV bude konstantní. Variací určitě naleznete více. Další poznámka se týká úpravy hesla. Obdržené bajty jsou v rozsahu hodnot 0 - 255, ale my potřebujeme hodnoty 0 - 9. Stačilo by třeba využívat číslo h(i) mod 10, ale v tom případě bychom nedostali jednotlivé číslice stejně pravděpodobné. Číslice 0 až 5 by byly oproti ostatním "ve výhodě", protože na ně vedou čísla 250 až 255, zatímco v ostatních desítkách jsou výskyty rovnoměrné. Proto pokud narazíme na bajty v rozsahu 250 - 255, ignorujeme je a žádnou číslici z nich nevytváříme. Tím ve skutečnosti upravujeme původní zdroj náhodných znaků z rozsahu 0 - 255 na zdroj náhodných znaků v rozsahu 0 - 245, a tedy po modulování na 0 - 9. Pokud by se jednalo o karetní aplikaci, kde potřebujeme čísla 0 - 31, stačí využívat z proudu hesla vždy dolních pět bitů, nebo šetřit a proud bajtů "sekat" rovnou po pěti bitech. Určitě vás napadne, že podobným způsobem lze z klíče ve formě textové fráze nebo passwordu (pokud je použijeme jako klíč nebo IV) podobným způsobem vygenerovat posloupnost znaků v jakémkoliv požadovaném rozsahu, třeba PIN apod.

Kód proudové šifry RC4

Příprava klíčové tabulky

Šifrovací klíč (zarovnaný na bajty) cyklicky vepisujeme do pole K(0),K(1), ..., K(255). Zvolíme identickou počáteční permutaci S, tj. S(i) = i, i = 0...255, a promícháme ji prostřednictvím hodnot K(i) podle následujícího pseudokódu takto (+ je sčítání v modulu 256):

```
j = 0
for i = 0 to 255 do
{
j = ( j + S(i) + K(i)) mod 256 v tabulce S vyměň hodnoty S(i) a S( j)
```

```
}
```

Generování hesla h(i)

```
x = y = 0
for i = 0 to n do
{
x = x + 1 y = y + S(x) v tabulce S vyměň hodnoty S(x) a S(y) h(i) = S ( S(x) + S(y) )
}
```

Šifrování

Jednotlivé bajty hesla h(i) je poté xorují na otevřený text (při zašifrování) nebo šifrový text (při dešifrování).